Authenticated Key Agreement Protocols: Security Models, Analyses, and Designs

Augustin Sarr

▶ **To cite this version:**

Augustin Sarr. Authenticated Key Agreement Protocols: Security Models, Analyses, and Designs. Mathematics [math]. Université Joseph-Fourier - Grenoble I, 2010. English. NNT : . tel-00532638v1

HAL Id: tel-00532638

https://theses.hal.science/tel-00532638v1

Submitted on 4 Nov 2010 (v1), last revised 18 Jan 2011 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE
Spécialité **Mathématiques et Informatique**

Arrêté ministériel : 7 août 2006

Présentée et soutenue publiquement par

Sarr Augustin P.

le **18 octobre 2010**

Protocoles d'échanges de clefs authentifiés : modèles de sécurité, analyses et constructions

---

Thèse dirigée par Philippe Elbaz-Vincent et codirigée par Jean-Claude Bajard

JURY

Jean–Claude Bajard	Prof. Université Paris 6	Examinateur
Cas Cremers	Senior Scientist, ETH Zurich	Examinateur
Philippe Elbaz–Vincent	Prof. Université de Grenoble	Examinateur
Xavier Facélina	Directeur Technique, Netheos	Examinateur
Louis Goubin	Prof. Université de Versailles–Saint–Quentin	Rapporteur
Antoine Joux	Prof. Université de Versailles–Saint–Quentin	Examinateur
Marc Joye	HDR, Technical advisor, Technicolor	Rapporteur
Yassine Lakhnech	Prof. Université de Grenoble	Examinateur

Thèse préparée au sein de l' Institut Fourier dans l' Ecole Doctorale MSTII

Université de Grenoble

Thèse

pour obtenir le grade de

Docteur
de l'Université de Grenoble

Formation Doctorale :	Mathématiques et Informatique
École Doctorale :	Mathématiques, Sciences et Technologies de l'Information, Informatique
Discipline :	Cryptologie

présentée et soutenue publiquement
par

Augustin P. Sarr

le 18 octobre 2010

Authenticated Key Agreement Protocols:
Security Models, Analyses, and Designs

—

Protocoles d'échanges de clefs authentifiés :
modèles de sécurité, analyses et constructions

— Jury —

Jean–Claude Bajard	Prof. Université Paris 6	Co–directeur
Cas Cremers	Senior Scientist, ETH Zurich	Examinateur
Philippe Elbaz–Vincent	Prof. Université de Grenoble	Co–directeur
Xavier Facélina	Directeur Technique, Netheos	Examinateur
Louis Goubin	Prof. Université de Versailles–Saint–Quentin	Rapporteur
Antoine Joux	DGA, Prof. Université de Versailles–Saint–Quentin	Examinateur
Marc Joye	HDR, Technical advisor, Technicolor	Rapporteur
Yassine Lakhnech	Prof. Université de Grenoble	Examinateur

Remerciements

Je tiens à exprimer ici ma profonde gratitude à tous ceux qui d'une façon ou d'une autre ont contribué à la réalisation de ce travail. En particulier, à mes directeurs de recherche Jean-Claude Bajard, Xavier Facélina et Philippe Elbaz-Vincent qui, après m'avoir initié aux différents aspects de la cryptologie et de la recherche, ont accepté de diriger ce travail de thèse. Ils m'ont aidé, conseillé et soutenu tout au long de ce travail; et très souvent, ils ont fait bien plus que leur travail. Nos différentes discussions ont inspiré l'essentiel des résultats présentés dans ce manuscrit. Je salue leurs qualités humaines. Qu'ils soient assurés de ma profonde reconnaissance.

Mes remerciement vont également à la société Netheos, en particulier à ses dirigeants Olivier Détour et Xavier Facélina qui ont aussi permis ce travail. Sans l'environnement favorable qui règne dans les locaux de cette équipe, ce travail m'aurait été très pénible; je remercie chaleureusement tous les salariés de Netheos.

Ma gratitude va également à Louis Goubin et Marc Joye qui m'ont fait l'honneur de rapporter sur ce travail. Je les remercie vivement pour l'effort qu'ils ont fourni et l'intérêt qu'ils ont porté à ce manuscrit. Mes vives remerciements vont également à Cas Cremers, Antoine Joux et Yassine Lakhnech qui ont bien voulu examiner ce travail et faire partie du jury.

Enfin, je ne saurais terminer ces remerciements sans en adresser à ma famille et à mes amis, qu'ils soient de Dakar, de Montpellier, de Mont-Rolland ou de Saint-Louis. Je leur dois beaucoup; qu'ils en soient remerciés.

À mon frère

Daniel SARR

Fey sosa faanu.

Contents

List of Algorithms and Protocols

List of Attacks

List of Figures

Notations

$\mathcal{G}$	A multiplicatively written cyclic group of prime order q generated by G.
$\bar{1}$	The identity element in $\mathcal{G}$.
$\mathcal{G}^*$	$\mathcal{G} \setminus \{\bar{1}\}$.
$\hat{A}$	An entity with public key A, when '$\hat{A}$' is used as input for some function, we refer to $\hat{A}$'s certificate.
$\mathcal{A}$	An attacker.
$Sign_{\hat{A}}(m)$	$\hat{A}$'s signature on a message m.
$Enc_K(m)$	Symmetric encryption of a message m using a key K.
$Dec_K(m)$	Decryption of a message m using a key K.
X	An element in $\mathcal{G}$.
x	The discrete logarithm of X in base G.
$\|x\|$	The bit length of x.
$\parallel k \parallel$	The absolute value of a number k.
(EC)DLP	Elliptic Curve Discrete Logarithm Problem.
CDHP	Computational Diffie–Hellman Problem.
ECDHP	Elliptic Curve Computational Diffie–Hellman Problem.
(EC)DDHP	(Elliptic Curve) Decisional Diffie–Hellman Problem.
(EC)GDHP	(Elliptic Curve) Gap Diffie–Hellman Problem.
$H, \bar{H}, H_{i,i\in\mathbb{N}}$	Digest functions.
KEA1	Knowledge–of–Exponent Assumption.
KDF	Key Derivation Function.
MAC	Message Authentication Code.
$\in_R$	"Chosen uniformly at random in".
$\{0,1\}^\lambda$	The set of binary stings of length λ.
$\{0,1\}^*$	The set of finite length binary stings.
$\mathbb{N}$	The set of non–negative integers, i.e. $\{0, 1, \cdots\}$.
$\mathbb{N}^*$	$\mathbb{N} \setminus \{0\}$.
$\gcd(n_1, n_2)$	The greatest common divisor of two integers n_1, n_2.
$GF(q)$	A finite field with q elements.
$(s_1, \cdots, s_n)$	The concatenation of $s_1, \cdots, s_n$, if $s_1, \cdots, s_n$ are (or can be represented) as binary strings.

Chapitre 1

Introduction (in french)

Contents

1.1 Contexte et motivation

Historiquement, la cryptographie a été souvent utilisée pour des besoins de *confidentialité*. Une entité désirant communiquer avec une autre, convient avec celle–ci d'un procédé de (dé)chiffrement ainsi que d'un secret. L'expéditeur qui souhaite son message confidentiel le chiffre avant de l'envoyer au destinataire. À la réception du message, le destinataire, qui a connaissance du procédé de déchiffrement ainsi que du secret à utiliser, applique le procédé et retrouve le message initial en clair. Cette approche de la confidentialité, dite *cryptographie symétrique* ou *cryptographie à clef secrète*, nécessite un accord préalable sur le secret de chiffrement. Il se pose donc le problème de la mise en œuvre de cet accord de façon à garantir la non–divulgation du secret, ainsi que celui du stockage des clefs, notamment lorsque les partenaires sont multiples.

Au besoin de confidentialité, que certains auteurs (voir [DRIO53], p. ex.) font remonter à l'Égypte antique, s'ajoutent les besoins fondamentaux d'authentification, d'intégrité des données et de signature. L'*authentification*, qui peut s'appliquer aussi bien aux entités qu'aux données, est la « garantie » de l'exactitude de la réalité d'une identité ou d'une information. L'*intégrité de données* a trait aux altérations de données ; le but étant, de rendre celles–ci au moins détectables. Une *signature* lie l'identité d'une entité à une donnée ; et comme telle, elle garantit l'origine de la donnée. Lorsque l'unicité d'un signataire est garantie, une signature vérifiable par tous induit (de par la garantie de l'origine) la non–répudiation, c.–à–d., l'incapacité d'un signataire à nier l'authenticité d'une donnée signée.

La démocratisation de l'accès à internet, ainsi que la dématérialisation des échanges (commerce en ligne, e–gouvernance, etc.) intensifient l'acuité des besoins fondamentaux, en plus d'en poser de nouveaux. Les exigences de sécurité et de performance, liées parfois des enjeux cruciaux, sont de plus en plus élevés.

Certaines limites de la cryptographie symétrique trouvent réponse dans l'approche révolutionnaire proposée par Diffie et Hellman[1] [DIF76]. Dans cette approche, chaque

[1] Il apparait que les services secrets britaniques avaient mis au point en 1973 un schéma de chiffrement à clef public, le GCHQ (voir http://www.gchq.gov.uk/history/pke.html). Par ailleurs, la relation entre fonctions à sens unique et cryptographie a été déjà évoquée, dans [JEV58] en 1874. La discussion portait spécifiquement sur la factorisation des entiers, aujourd'hui à la base du fameux schéma RSA.

entité dispose d'un secret (dit *clef privée*) lié à une information publique (*clef publique*). Le chiffrement à destination d'une entité se fait en utilisant sa clef publique, le déchiffrement nécessite la clef privée. Le besoin d'accord préalable sur un secret de chiffrement partagé ne se pose donc plus, tout comme celui de la multiplication des secrets à stocker pour de multiples partenaires. Les clefs publique et privée sont liées par une relation connue ; cependant, pour une paire de clefs « bien choisie », il doit être infaisable de calculer la clef privée à partir de la partie publique.

En pratique, la question de la distribution et de l'authenticité des clefs publiques est traitée via des *infrastructures à cléfs publiques*. Le principe d'une telle infrastructure consiste à faire partager à des entités, une entité de confiance, dite *autorité de certification* qui signe (donc lie), après validation, une clef publique et une identité. L'objet ainsi signé est dit *certificat*.

Bien qu'apportant des réponses à certaines limites de la cryptographie à clef secrète, en pratique la cryptographie asymétrique est moins performante. En effet, à niveau de sécurité équivalent, les schémas de chiffrement à clef publique les plus courants sont nettement plus lents que les schémas symétriques. Afin de mitiger ce manque relatif de performance, les schémas asymétriques et symétriques sont généralement utilisés de paire. Lorsque les données à émettre sont de taille conséquente, une combinaison possible (probablement la plus simple), d'un schéma symétrique et asymétrique est comme suit :

- L'émetteur $\hat{A}$ choisit une clef secrète s au hasard, chiffre la clef secrète avec la clef publique du destinataire, ici $\hat{B}$, puis, chiffre son message m avec la clef secrète.
- Le destinataire, utilisant sa clef privée b, déchiffre la clef secrète et, utilisant cette dernière, accède au message m.

$$
\begin{array}{lll}
\underline{\hat{A}} : \hat{B} & & \underline{\hat{B}} : b, B, \hat{B} \\
s \in_R \{0,1\}^k, & & \\
c_s = Enc_B(s), & & \\
c_m = Enc_s(m), & & \\
 & \boxed{c_m, c_s} \longrightarrow & \\
 & & s = Dec_b(s), \\
 & & m = Dec_s(c_m),
\end{array}
$$

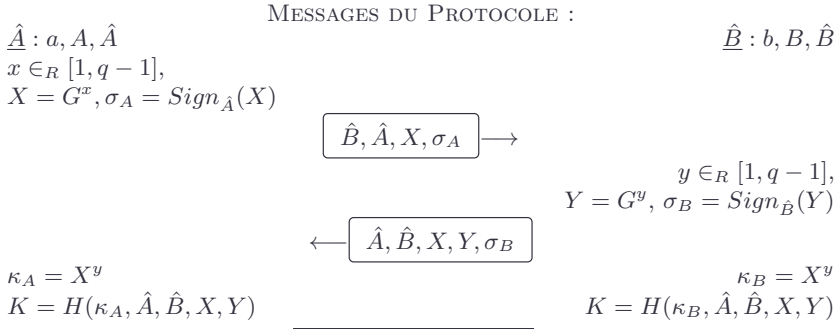
Dans cette combinaison, (1) une des entités choisit la clef de chiffrement du message (clef de session) et la fait parvenir à son partenaire ; (2) la communication suivante (ici de l'initiateur vers le destinataire) utilise la clef choisie par l'initiateur. Les protocoles construits avec cette idée, qui consiste à faire choisir une clef de session par une des entités et la faire parvenir aux autres, sont dits *protocoles de transport de clef*.

Bien qu'efficace, cette approche n'est pas adaptée à certaines situations. En effet, il peut être désirable pour une entité d'avoir des garanties sur la *fraîcheur* des clefs de session qu'elle utilise. Les motivations de cet objectif de fraîcheur peuvent être de deux origines. La première concerne la divulgation potentielle des clefs session : les clefs peuvent être utilisées dans des applications ou espaces de stockage qui ne sont pas de confiance, le format et le volume des données chiffrées peut les rendre

vulnérables à une cryptanalyse, etc. La seconde source de motivation, sans doute la plus importante, est qu'il est souhaitable que les clefs sessions soient décorrélées de façon à éviter le rejeu d'un message d'une session antérieure, par une entité malicieuse. Il peut être donc souhaitable pour chaque entité d'avoir la garantie que la clef de session qu'elle utilise est fraîche, et n'a fait l'objet d'aucune utilisation antérieure. En exemple, pour les protocoles d'authentification en temps–réel, il est souvent désirable de pouvoir garantir qu'une entité malicieuse ne peut rejouer les messages d'une session antérieure. Globalement, pour une entité, la garantie de la fraîcheur d'une donnée peut être obtenues de deux manières : (1) l'entité participe directement à la génération de la donnée, ou (2) la donnée, alors reçue, est liée à une donnée fraîche (dite *nonce*) fournie par l'entité en question.

Dans un protocole d'échange de clef, il est commun que la dérivation d'une clef de session, en une entité, implique à la fois des données fraîches générées par l'entité, ainsi des données reçues d'une entité distante. De plus, les données reçues peuvent être liées (de façon implicite ou explicite) à des données fraîches.

Protocole 1.1 Variante à deux messages du protocole signé de Diffie–Hellman



I) L'initiateur $\hat{A}$ effectue les étapes suivantes
   (a) Choisir $x \in_R [1, q-1]$, calculer $X = G^x$ et $\sigma_A = Sign_{\hat{A}}(X)$.
   (b) Envoyer $(\hat{B}, \hat{A}, X, \sigma_A)$ à $\hat{B}$.

II) À la reception de $(\hat{B}, \hat{A}, X, \sigma_A)$, $\hat{B}$ effectue les étapes suivantes :
   (a) Vérifier que $X \in \mathcal{G}^*$ et valider la signature σ_A.
   (b) Choisir $y \in_R [1, q-1]$, calculer $Y = G^y$ et $\sigma_B = Sign_{\hat{B}}(Y)$.
   (c) Envoyer $(\hat{A}, \hat{B}, X, Y, \sigma_B)$ à $\hat{A}$.
   (d) Calculer $\kappa_B = X^y$.
   (e) Calculer $K = H(\kappa_B, \hat{A}, \hat{B}, X, Y)$.

III) À la réception de $(\hat{A}, \hat{B}, X, Y, \sigma_B)$, $\hat{A}$ effectue les étapes suivantes :
   (a) Vérifier que $Y \in \mathcal{G}^*$ et valider la signature σ_B.
   (b) Calculer $\kappa_A = Y^x$.
   (c) Calculer $K = H(\kappa_A, \hat{A}, \hat{B}, X, Y)$.

IV) La clef de session partagée est K.

En présence d'une infrastructure à clef publiques, il apparaît attractif de signer les messages du protocole Diffie–Hellman (dont les limites, notamment pour ce qui est de l'attaque dite « de l'homme du milieu » sont bien connues). Une variante possible du protocole résultant est décrite ci–dessus (Protocole 1.1).

Le Protocole 1.1 satisfait un certain nombre d'attributs de sécurité. Si l'on admet l'hypothèse qu'une entité malicieuse ne saurait accéder aux exposants privés éphémères x et y, il est défendable que l'objectif d'*authentification mutuelle* est satisfait, sous l'hypothèse du problème calculatoire de Diffie–Hellman. En effet, sous cette hypothèse, comme seul $\hat{B}$ peut générer une signature en son nom, pour toute signature validée par $\hat{A}$, seul $\hat{B}$ peut avoir connaissance de l'exposant de la clef privée éphémère, et inversement. Ainsi, seuls $\hat{A}$ et $\hat{B}$ peuvent calculer $\kappa_A = \kappa_B$ et dériver la clef de session. Il peut aussi être désirable pour $\hat{A}$ d'avoir la « preuve » que $\hat{B}$ dispose effectivement de la clef de session (ou inversement) ; cet objectif est dit *confirmation de clef*. Il n'est pas difficile de voir que le Protocole 1.1 ne satisfait pas cet objectif.

Pour Shoup [SHO99] la confirmation de clef n'est pas importante, seuls importent l'information sur l'*aboutissement d'une session partenaire* et l'*authenticité et la confidentialité de la clef de session*. Autrement dit, pour une session entre $\hat{A}$ et $\hat{B}$, pour $\hat{A}$, seul importe le fait que la sesssion en $\hat{B}$ aboutisse, et la garantie qu'aucune entité en dehors de $\hat{A}$ et $\hat{B}$ ne peut calculer la clef de session. Nous ne discuterons pas de l'importance ou non de la confirmation de clef en terme de sécurité. On peut juste remarquer que d'un point de vue fonctionnel, il peut être important ; en example, avant le chiffrement d'un gros volume de données, il peut être souhaitable de s'assurer que le destinataire peut effectivement réussir le déchiffrement, et s'éviter ainsi un effort de chiffrement inutile.

La démarche de construction et d'évaluation des protocoles cryptographiques a souvent similaire à celle que nous venons d'éffectuer avec le Protocole 1.1 ; les protocoles sont construits sur la base de l'intuition, et analysés de façon informelle. Un schéma étant considéré comme adéquat, s'il résiste à la cryptanalyse au bout d'un certain nombre d'années. Cette approche a conduit à un nombre conséquent de protocoles, dont une vaste majorité de designs qui se sont révélés insuffisants en terme de sécurité.

Il est à noter (à juste titre) que l'hypothèse de l'adversaire passif, faite ci–dessus, ne saurait correspondre à la réalité de la majorité des environnement d'implantation. De plus, le propre d'un attaquant étant de mettre un système dans un état autre que ceux initialement prévus par le designer, notre hypothèse sur l'environnement, tout comme nos arguments informels sur la sécurité du Protocole 1.1 ne sauraient suffire, notamment eu égard aux conséquences potentielles d'une défaillance du protocole en environnement de déploiement. Il est aussi à observer que toute divulgation d'un exposant privé éphémère d'une session ayant abouti en $\hat{A}$ donne à l'attaquant la possibilité de se faire passer *indéfiniment* pour $\hat{A}$ auprès de n'importe quelle autre entité. L'objectif qu'un attaquant ne devrait pas pouvoir se faire passer pour une entité à moins d'avoir connaissance de la clef privée statique de celle–ci n'est pas satisfait.

Un nouvelle approche dans la l'analyse des protocoles cryptographiques a été introduite par Bellare et Rogaway [BEL93a]. Cette nouvelle méthode adapte la démarche d'analyse des algorithmes cryptographiques de Goldwasser et Micali [GOL84], à l'analyse des protocoles d'échange de clef. La démarche, dite de « sécurité prouvée », consiste : (1) à définir avec précision les objectifs de sécurité du protocole à construire,

(2) à construire un protocole « candidat », et (3) à lier la sécurité du protocole ainsi défini à celui d'un problème admis difficile. Pour cela, il est souvent montré qu'un adversaire qui réussit à violer la sécurité du protocole, peut être utilisé pour résoudre de façon efficace le problème admis difficile.

La terminologie utilisée est cependant quelque peu controversée ; en effet, pour certains auteurs, [KOB07a, KOB07b] notamment, le terme « sécurité prouvée » prête à confusion, d'autant plus que des limites sont connues sur certaines définitions et réductions de sécurité, et peut être avantageusement remplacé par « sécurité par réduction ».

La sécurité d'un protocole n'est pas simplement à considérer au regard des hypothèses qui conduisent à la réduction, mais aussi au regard de la qualité du modèle dans lequel les arguments de sécurité sont établis. L'adéquation du modèle à la réalité des environnements d'implantation et de déploiement est aussi à importante.

Depuis les travaux initiateurs de Bellare–Rogaway [BEL93a], les modèles de sécurité pour l'analyse des protocoles n'ont cessé d'évoluer ; la finesse des définitions est souvent améliorée. Bien que les objectifs pratiques de sécurité soient pour l'essentiel les mêmes, les définitions de sécurité proposées ne sont pas toujours formellement comparables. Parmi les modèles proposés jusque là, les modèles dits de Canetti–Krawzyk [CAN01] et Canetti–Krawzyk étendu [LAMA07] sont considérés comme étant les plus évolués. Pour autant, un regard attentif sur certains protocoles montrés sûrs dans ces modèles suggère une inadéquation entre l'analyse formelle et la réalité à laquelle est souvent confronté l'implanteur d'un protocole. Cette thèse, traite à la fois de l'analyse des définitions de sécurité, de Canetti–Krawzyk et de Canetti–Krawzyk étendu notamment, de l'analyse et de la construction des protocoles d'échange de clef authentifiés.

1.2 Contributions

Nous montrons que les définitions de sécurité de Canetti–Krawzyk [CAN01] et Canetti–Krawzyk étendu [LAMA07] présentent des subtilités qui font que certaines attaques, qui peuvent être menées en pratique, ne sont pas considérées dans les analyses de sécurité. Nous illustrons ces limites avec attaques sur des protocoles montrés sûrs dans ces modèles de sécurité.

Nous proposons une forte définition de sécurité, qui englobe le modèle eCK, et prend en compte des aspects pratiques liés à l'implantation des protocoles. Nous proposons une analyse complémentaire des schémas de signature XCR (“Exponential Challenge Response”) et DCR (“dual exponential Challenge Response”), qui sont les briques du protocole HMQV. Nous introduisons de nouveaux points, dits i–points, qui peuvent être utilisés lorsque leur décomposition est connue pour une attaque par impersonation contre les protocoles (C, H)MQV(–C). Nous explorons la recherche de ces points et de leur décomposition, et montrons qu'elle peut s'effectuer environs deux fois plus vite que la résolution du problème du logarithme discret. Nous expérimentons la recherche de i–points décomposés sur des courbes de taille réduite, l'expérimenta-

tion confirme les avantages de notre approche. Sur la base de cette analyse, nous montrons la vulnérabilités des protocoles (C, H)MQV(–C) [UST08, LAW03, KRA05b] aux fuites d'informations spécifiques à une session. Nous montrons notamment que lorsqu'un attaquant accède à certaines informations de session, qui ne conduisent pas à une divulgation de la clef statique du détenteur de la session, il peut réussir une attaque par usurpation d'identité.

Nous proposons les schémas de signature FXCR ("Full XCR") et FDCR ("Full DCR") à partir desquels nous construisons les protocoles FHMQV ("Fully Hashed MQV") et SMQV ("Strengthened MQV") qui préservent la performance remarquable des protocole (H)MQV, en plus d'une meilleure résistance aux fuites d'informations. Les protocoles FHMQV et SMQV sont particulièrement adaptés aux environnements dans lesquels une machine non digne de confiance est combinée avec un module matériel à faible capacité de calcul et résistant aux violations physiques de sécurité. Dans un tel environnement, les opérations effectuées sur le module matériel hors temps mort se réduisent à des opérations peu coûteuses. Nous montrons que les protocoles FHMQV et SMQV satisfont notre définition de sécurité sous les hypothèses de l'oracle aléatoire et du problème échelon de Diffie–Hellman.

Le travail présenté dans ce manuscrit a bénéfiée d'un financement Cifre. Il est aussi le fruit de la la collaboration entre la société Netheos et des laboratoires I3M, Institut fourier, LIRMM, à travers notamment les professeurs Jean–Claude Bajard et Philippe Elbaz–Vincent.

1.3 Plan du manuscrit

Le chapitre 2 donne un aperçu globale des courbes elliptiques et de leur utilisation en cryptographie. Nous rappelons brièvement les courbes elliptiques, l'arithmétique afférente dans le cas des corps finis, ainsi que quelques schémas (qui peuvent être) basés sur les courbes elliptiques.

Dans le chapitre 3 nous rappelons les modèles de Bellare–Rogaway, de Canetti–Krawzyk (CK) [CAN01] et le modèle dit de Canetti–Krawzyk étendu (eCK). Nous illustrons les limites de ces modèles et proposons le modèle seCK (*strengthened eCK*). La relation du modèle seCK au modèles (e)CK [LAMA07] est explorée également. Afin d'illustrer le fait que le modèle seCK n'est pas très restrictif, nous proposons les schémas d'identification FXCR–1 et FDCR–1, avec lesquels nous construisons le protocole SMQV qui satisfait la seCK sécurité sous les hypothèse du problème calculatoire de Diffie–Hellman et de l'oracle aléatoire.

Dans le chapitre 4, nous introduisons les i–points et explorons la relation de ces points aux attaques par impersonation. Nous discutons des facilités de parallélisation liées à la recherche de ces points, et montrons cette recherche, peut être combinée à l'algorithme rho de Pollard. Nous montrons certaines limites des protocoles (C, H)MQV(–C), et proposons le protocole FHMQV, que nous montrons sûr dans le modèle seCK.

Le chapitre 5 présente le standard PKCS #11 ; nous discutons les limites du standard, et des restrictions qui peuvent être nécessaires pour une implantation sécurisée. Nous proposons une discussion succincte sur certains aspects des implantations que nous avons eu à effectuer. La discussion relative aux implantations, qui sont aujourd'hui utilisées dans des produits commerciaux, est volontairement limitée.

Enfin, dans le chapitre 6, nous suggérons des pistes à explorer pour des travaux futures.

Introduction

Key exchange protocols are fundamental elements in network communications security. A key exchange protocol is said to be authenticated if each implicated entity is assured that no other entity, but those identified can compute the shared key. Broadly, following their design elements, authenticated key exchange protocols can be divided into two groups: those in which authentication is achieved via explicit signatures, and those in which authentication is implicitly guaranteed by the ability of implicated parties to compute the shared key. The later has attracted more interest because protocols with implicit authentication are generally more efficient than the others. Even if the concept of a secure protocol may seem intuitive, a rigorous formalization of this notion is notoriously far from being a simple task [BOY03, chap .2].

Moreover, security is meaningless, except in reference with a well defined security model. A security model specifies, among other things, what constitutes a security failure, and what adversarial behaviors are being protected against. The aim is that a protocol shown secure in the model, confines to the minimum the effects of the considered adversarial behaviors.

Besides the considered definition, security must be understood in regard to the assumptions under which the arguments are given. Proving that a cryptographic protocol is secure is a subtle task; there are many technicalities and possible interactions involved. For a secure protocol, it should be infeasible for an adversary, eavesdropping or altering communications between parties, to make the protocol fail in any of its security goals. In particular, for authenticated key exchange protocols, it should be impossible for an attacker in control of communications between parties, to impersonate a party, unless it knows the party's static key. Designing a secure key exchange protocol is also a difficult task. Most of the proposed protocols have only heuristic arguments; and with the benefit of hindsight, many of the protocols previously claimed provably–secure, turn out to be flawed, or designed with reference to a security model which is not strong enough.

Since the pioneering complexity theoretic work of Bellare and Rogaway [BEL93a], the foundations of the definitions of a secure key exchange seem well established. Based on [BEL93a], different security definitions [BEL95, BLA97a, SHO99, CAN01, KRA05, LAMA07] were proposed. Even if the definitions are not always formally comparable [CRE09b, CRE09a, CHO05] they use the same fundamental approach. A protocol is secure if an adversary controlling communications between parties, cannot distinguish a session key from a random value chosen under the distribution of session keys, unless it makes queries which overtly reveal the session key. Among the models used in the analysis of authenticated key agreement protocols, the Canetti–Krawczyk

(CK) [CAN01] and extended Canetti–Krawczyk (eCK) models [LAMA07] are considered as "advanced" security definitions to capture security for key agreement protocols. Security arguments for recent protocols are usually provided in these models. Unfortunately, even if the (e)CK models are considered as advanced security definitions, there remains unconsidered practical attacks, which can make part of the protocols shown secure in these models fail in their fundamental security goals in practice.

Furthermore, while current security definitions are used for protocols security arguments, they are of no help in protocol design. There is no well established paradigm for the design of authenticated key agreement protocols. Analyzing the building blocks of already well–established efficient schemes, in order to identify the design choices which make them more efficient than the others is also important.

In this dissertation, we consider the security definitions in regard to which much of the recent protocols security arguments are provided. We point out some limitations in the Canetti–Krawczyk and extended Canetti–Krawczyk models which make some practical attacks unconsidered in security arguments. We also propose a new security model which encompasses extended Canetti–Krawczyk model, and captures the intuition of a secure protocol implementation. We also propose two efficient protocols, which provably meet our security definition, under the Random Oracle model and the Gap Diffie–Hellman Assumption. The protocols we present are particularly suited for distributed implementation environment wherein a computationally limited tamper resistant device is used, together with an untrusted host machine. In such environments, for our protocols, the non–idle time computational effort of the device safely reduces to few non–costly operations.

This dissertation is organized as follows. In chapter 2, we present an overview of elliptic curves based cryptography, and related industrial problematics. Even if our results are presented in a generic group, this dissertation is mainly written with elliptic curve groups in mind (and even jacobian of hyperelliptic curves).

In chapter 3, we outline the original Bellare–Rogaway (BR) model, and present the (e)CK models. We also highlight the importance of finely understanding the limitations of the (e)CK models, when using them in security reductions. We show how the CK matching sessions definition makes some practical impersonation attacks unconsidered in security arguments. We also show how the use of the NAXOS transformation [LAMA07] leads to protocols which are formally eCK, but also practically insecure. We propose a strong security definition which encompasses the eCK model, and provides stronger reveal queries to the adversary. We also propose a new authenticated key agreement protocol called Strengthened MQV (SMQV), which meets our security definition under the gap Diffie–Hellman assumption and the Random Oracle model.

In chapter 4, we illustrate the two main approaches for achieving authentication in Diffie–Hellman protocols. We do so using a enhanced variant of the UM protocol from [NIS07], and variants of the Station–to–Station protocol. After that, we restrict our attention on the design elements of the famous MQV and HMQV protocols, which are probably the most efficient of all authenticated Diffie–Hellman protocols. We pro-

pose a complementary analysis of the Exponential Challenge Response (XCR) and Dual Exponential Challenge Response (DCR) signature schemes. On the basis of this analysis we show how impersonation and man in the middle attacks can be performed against the (C, H)MQV protocols when some session specific information leakages occur. We propose the Full Exponential Challenge Response (FXRC) and Full Dual Exponential Challenge Response (FDCR) signature schemes. Using these schemes we define the Fully Hashed MQV (FHMQV) protocol, which resists the attacks we present and preserves the remarkable performance of the (H)MQV protocols.

In chapter 5, we propose an analysis of the Public Key Cryptography Standards (PKCS) #11 standard specification and its implementations. We discuss sensitive keys export, key space reduction, key wrapping based fault attacks, and more generally the security consequences of allowing conflicting security attributes in PKCS #11 objects. Finally, we present few of the technicalities related to our implementations of PKCS #11, concerning the Everbee SMK ("Smart Mobile Key") and the Reconfigurable Cryptographic Platform (RCP). The discussion in this chapter is voluntarily limited, because of the commercial nature of the involved products and implementations.

Chapter 2

Elliptic Curve Cryptography and Related Industrial Problematics

Contents

2.1 Introduction

Computer security and more generally network security involves many security facets, physical security (tamper protection mechanisms), logical security, environmental security, etc. In practice, many security mechanisms are conjointly used to fulfill practical needs in a given context.

Historically cryptography has been used for *confidentiality*, a message sender encrypts the message, and transmits the encrypted text. The receiver decrypts the encrypted message to get the content. Naturally, it is necessary that the message sender and receiver previously agree on an encryption and decryption process, and a particular key. Such cryptographic schemes are termed *symmetric cryptography*. Symmetric cryptography comes with the natural concern of secure key sharing. As computing systems are today highly interconnected, there is also a need for remote systems to identify one another with a high degree of confidence; this need is often coupled with that of key sharing.

Another encryption approach was proposed in seventies [DIF76]; in this approach different keys are used for encryption and decryption. The encryption key, termed *public key*, is publicly available, while the decryption key (*private key*) is kept secret by the owner. As the two keys are related in some publicly known way, a knowledge of the public key allows *in theory* to compute the private part. However, for well defined schemes, it is *computationally infeasible* to compute the private part from the public one. Such schemes (and related techniques) are termed *public key cryptography*. Public key schemes only are not sufficient for security; a public key must be bind to its owner's identity. In practice this binding is commonly performed using a *public key infrastructure*. The principle of a public key infrastructure consists in making a third party, trusted by other parties, provide this binding through signatures. The signed data for party, which includes the party's identity and public key is termed *certificate*. A main benefit of public key encryption is that any party which knows the public key of the receiver can securely send to it a message, while in symmetric key cryptography, only a set of parties sharing a secret key can securely communicate. However, public key schemes seem to be slower than the symmetric ones. In practice, symmetric schemes are often used to encrypt a large amount of data, while public key schemes are used for key distribution and establishment. Also for public, public key infrastructures infrastructures are needed for public key distribution.

Cryptography has gone beyond its traditional use for protecting information transmission. It fulfills many needs in modern information systems; using cryptography, the following can be achieved [MEN96, chap. 1].

(a) *Confidentiality*, which is keeping information secret from all parties, but those allowed to see it; *secrecy* and *privacy* are also used as synonyms of confidentiality.

(b) *Authentication* applies to both entities and information itself; it is the guarantee that a claimed identity or a given information is authentic.

(c) *Data integrity* addresses the unauthorized alteration of data. To insure data integrity, tampering which includes data insertion, deletion, and substitution must be detected.

(d) *A signature* binds the identity of an entity with some information, it provides *non–repudiation*, i.e, the inability to deny the authenticity of the information. Only information sender can generate a valid signature which binds its identity to the information, while anyone can verify the validity of a given signature.

In this chapter, we provide an overview of the use of elliptic curves in cryptography. We provide a brief introduction to elliptic curves, and (related) discrete logarithm problem(s); few elliptic curve based schemes are discussed to illustrate the use of elliptic curves. Finally we provide an overview of standards related to elliptic curve cryptography, and rapidly discuss some patent related issues.

2.2 Overview of Elliptic Curves

The use of elliptic curves in cryptography was independently proposed by Koblitz [KOB87] and Miller [MIL86]. Since then, many research deals with the design and analysis of elliptic curve based schemes; applications of elliptic curves include also,

among others, integer factorization and primality proving [LEN87, GOL84].

The number–theoretic problems upon which are based the most commonly used public–key schemes are: (1) the integer factorization problem, upon which are based the well–known RSA encryption and signature schemes; (2) the multiplicative discrete logarithm problem over finite fields, used for the design of the Elgamal and DSA signature schemes [MEN96, chap. 11]; and (3) the elliptic curve discrete logarithm problem, which hardness is crucial for the elliptic curve based schemes.

The topic of elliptic curves involves a large amount of mathematics; our aim in this section is to summarize the basic theory for cryptographic needs. We briefly recall the elliptic curves, and their group law. For additional readings, a concise treatment can be found in [HAN03], more detailed treatments can be found in [SIL86, SIL92, WAS08, COH05b, BLA00].

Let K be a field, the two–dimensional *projective space* $\mathbb{P}^2(K)$ is the equivalence classes of the triples (x, y, z), with $x, y, z \in K$ and not all null; (x, y, z) and (x', y', z') are said to be equivalent if there is some $\lambda \in K$ such that $(x, y, z) = (\lambda x', \lambda y', \lambda z')$, their equivalence class is denoted $(x : y : z)$. If $(x : y : z) \in \mathbb{P}^2(K)$ with $z \neq 0$, then $(x : y : z) = (x/z : y/z : 1)$; these points are said to be "finite." The points $(x : y : 0)$ are termed *points at infinity* in $\mathbb{P}^2(K)$. Let $\mathbb{A}^2(K) = \{(x, y) \in K \times K\}$ be the two–dimensional *affine plane* over K, we have an inclusion $\mathbb{A}^2(K) \hookrightarrow \mathbb{P}^2(K)$ given by $(x, y) \longmapsto (x : y : 1)$.

A multivariate polynomial $P(x_1, \cdots, x_k)$ is said to be homogeneous of degree d, if for all $(x_1, \cdots, x_n)$, $P(\lambda x_1, \cdots, \lambda x_k) = \lambda^d P(x_1, \cdots, x_k)$. Hence, if $F(x, y, z)$ is homogeneous, $(x : y : z)$ and $(x' : y' : z')$ in $\mathbb{P}^2(K)$ such that $(x : y : z) = (x' : y' : z')$ and $F(x, y, z) = 0$, then $F(x', y', z') = 0$. And, if $f(x, y)$ is a polynomial of degree d, we have the homogeneous polynomial $F(x, y, z) = z^d f(x/z, y/z)$; the zeros of F in $\mathbb{P}^2(K)$ is the set of $(x_0 : y_0 : 1)$ such that (x_0, y_0) is a zero of $f(x, y)$ added with the zeros at infinity. Conversely, if $F(x, y, z)$ is an homogeneous polynomial of degree d, and $(x : y : z) \in \mathbb{P}^2(K)$ with $z \neq 0$ then $F(x, y, z) = z^d f(x/z, y/z)$ and $f(x, y) = F(x, y, 1)$.

Definition 1 (Elliptic Curve [HAN03, chap. 3])**.** Let K be a field and $\bar{K}$ its algebraic closure. An elliptic curve E is the set of zeros in $\mathbb{P}^2(\bar{K})$ of an homogeneous polynomial

$$F(x, y, z) = y^2 z + a_1 xyz + a_3 yz^2 - x^3 - a_2 x^2 z - a_4 xz^2 - a_6 z^3$$

where $a_1, a_2, a_3, a_4, a_6 \in K$, and F being non–singular; namely, there is no $(x_0 : y_0 : z_0) \in \mathbb{P}^2(\bar{K})$, such that $\dfrac{\partial F}{\partial x}(x_0, y_0, z_0) = \dfrac{\partial F}{\partial y}(x_0, y_0, z_0) = \dfrac{\partial F}{\partial z}(x_0, y_0, z_0) = 0$.
The affine variant of E is given by the polynomial

$$f(x, y) = y^2 + a_1 xy + a_3 y - x^3 - a_2 x^2 - a_4 x - a_6.$$

The quantity $\Delta = -b_2^2 b_8 - 8b_4^3 - 27b_6^2 + 9b_2 b_4 b_6 \neq 0$, wherein $b_2 = a_1^2 + 4a_2$, $b_4 = 2a_4 + a_1 a_3$, $b_6 = a_3^2 + 4a_6$, and $b_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2$ is said to be the *discriminant.*

If L is any extension of K, the *set of rational points* of E over L is

$$E(L) = \Big\{(x : y : z) \in \mathbb{P}^2(K) : \ y^2z + a_1xyz + a_3yz^2 - x^3 - a_2x^2z - a_4xz^2 - a_6z^3 = 0\Big\}.$$

A projective rational point $(x : y : z)$ with $z \neq 0$ corresponds to the affine point $(x/z, y/z)$. The projective point $(0 : 1 : 0)$ is also rational point, somewhat abusively termed ∞ in both the projective and affine representations. When rational points are represented in affine coordinates, $E(L)$ is equivalent to

$$\Big\{(x, y) \in K \times K : \ y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\Big\} \cup \{\infty\}$$

wherein ∞ corresponds to the projective point $(0 : 1 : 0)$.
The quantities b_2, b_4, b_6, and b_8 are only used to shorten the definition of Δ; the condition $\Delta \neq 0$ ensures that the curve E has no *singularity*.

Simplified equations. Two elliptic curves over a field K, given by

$$E_1 : \ \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

and

$$E_1 : \ y^2 + a_1'xy + a_3'y = x^3 + a_2'x^2 + a_4'x + a_6'$$

are said to be *isomorphic* over K if there are some $u \neq 0, r, s, t \in K$, such that $(x, y) \longmapsto (u^2x + r, u^3y + u^2sx + t)$ transforms E_1 into E_2. Such a transformation is said to be an *admissible change of variables.*
Let $E : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ be an elliptic curve over K, and $char(K)$ the characteristic of K.

(1) If $char(K) \neq 2, 3$, the admissible change of variables $(x, y) \longmapsto \Big(\dfrac{x - 3a_1^2 - 12a_2}{36}, \dfrac{y - 3a_1x}{216} - \dfrac{a_1^3 + 4a_1a_2 - 12a_3}{24}\Big)$ leads to an isomorphic curve $E' : \ y^2 = x^3 + ax + b$, with $a, b \in K$, and with discriminant $\Delta_{E'} = -16(4a^3 + 27b^2)$.

(2) If $char(K) = 2$, and $a_1 \neq 0$, then $(x, y) \longmapsto \Big(a_1^2x + \dfrac{a_3}{a_1}, a_1^3y + \dfrac{a_1^2a_4 + a_3^2}{a_1^3}\Big)$ yields the isomorphic curve $E' : \ y^2 + xy = x^3 + ax^2 + b$, with $a, b \in K$; E' is said to be *non–supersingular*, its discriminant is $\Delta_{E'} = -b$.
If $a_1 = 0$, the admissible change of variables $(x, y) \longmapsto (x + a_2, y)$ yields the curve $E' : \ y^2 + cy = x^3 + ax + b$, where $a, b, c \in K$; E' is *supersingular*, and has discriminant $\Delta = c^4$.

(3) If $char(K) = 3$, and $a_1^2 \neq -a_2$, $(x, y) \longmapsto \Big(x + \dfrac{a_4 - a_1a_3}{a_1^2 + a_2}, y + a_1x + \dfrac{a_1(a_4 - a_1a_3)}{a_1^2 + a_2} + a_3\Big)$ leads to $E' : \ y^2 = x^3 + ax^2 + b$, with $a, b \in K$. E' is *non–supersingular*and its discriminant is $\Delta = -a^3b$.
If $a_1^2 = -a_2$, then $(x, y) \longmapsto (x, y + a_1x + a_3)$ leads to the isomorphic curve $E' : \ y^2 = x^3 + ax + b$, with $a, b \in K$; E' is supersingular and has discriminant $\Delta = -a^3$.

Elliptic curves in Weierstrass form can be simplified using admissible transformations; in the continuation, we suppose curves represented in their simplified forms.

Group Law. Let E be an elliptic curve over a field K, $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ two finite K–rational points. The line joining P and Q is a rational line (i.e., a line with coefficients in K), and it generally meets E in a third point R'. If so, as the intersections of E and the rational line are given by the solutions of a cubic equation with rational coefficients, and P and Q are rational, R' is rational also. The sum of P and Q is defined to be the reflexion of R' on the x–axis, R in Figure 2.1. To double a point P, one first draw the tangent line to the elliptic curve at P, this line intersects the curve at a second point R'; the double R is the reflection of R' across the x–axis.

The set of rational points of an elliptic curve over a field K can be endowed with an abelian group structure; even if any of the rational points can be used as the identity element, the point at infinity ∞ is commonly used for this. The group law is described hereunder in affine coordinates for simplified equations over finite fields of characteristic different from 2 and 3, and also for curves over binary fields.

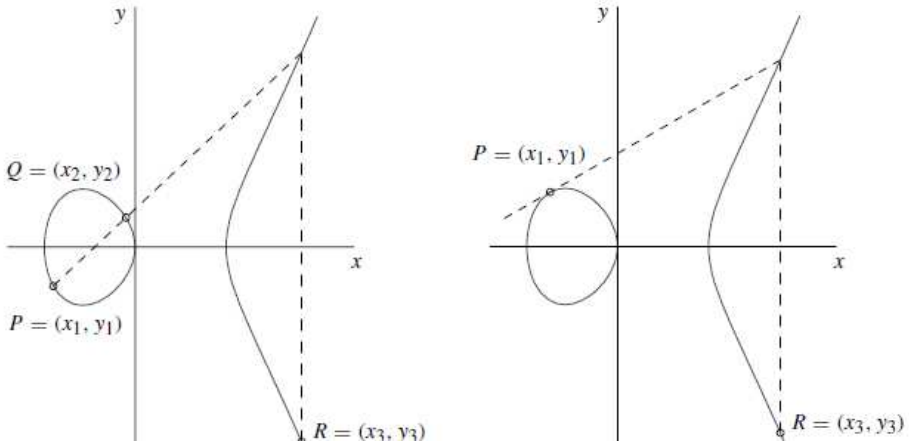


Figure 2.1: Chord-and-tangent rule

If E is an elliptic curve defined over a field K of characteristic $char(K) \neq 2, 3$, and given by $E: \; y^2 = x^3 + ax + b$, with $a, b \in K$, the group law is as follows.

- For all $P = (x_1, y_1) \in E(K)$, $P + \infty = \infty + P = P$; the opposite of P is $-P = (x_1, -y_1)$.
- If $Q = (x_2, y_2) \in E(K)$ and $Q \neq \pm P$, the sum of P and Q is $P + Q = (x_3, y_3)$ where $x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2$ and $y_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)(x_1 - x_3) - y_1$.
- The double of P is given by $2P = (x_3', y_3')$ where $x_3' = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$, and $y_3' = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$.

If E is a non–supersingular curve over a field $GF(2^m)$, given by $E: \; y^2 + xy = x^3 + ax^2 + b$, $a, b \in K$

- For all $P = (x_1, y_1) \in E(K)$, $P + \infty = \infty + P = P$; the opposite of P is $-P = (x_1, x_1 + y_1)$.
- If $Q = (x_2, y_2) \in E(K)$ and $Q \neq \pm P$, $P + Q = (x_3, y_3)$, where $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ and $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$, wherein $\lambda = \dfrac{y_1 + y_2}{x_1 + x_2}$.
- The double of $P \neq -P$ is $2P = (x'_3, y'_3)$ where $x'_3 = \lambda^2 + \lambda + a$ and $y'_3 = x_1^2 + \lambda x_3 + x_3$, wherein $\lambda = \dfrac{y_1}{x_1} + x_1$.

If E is a supersingular curve over $GF(2^m)$, given by $E : y^2 + cy = x^3 + ax^2 + b$, with $a, b, c \in K$

- For all $P = (x_1, y_1) \in E(K)$, $P + \infty = \infty + P = P$, and $-P = (x_1, y_1 + c)$.
- If $Q = (x_2, y_2) \in E(K)$ and $Q \neq \pm P$, the sum of P and Q is $P + Q = (x_3, y_3)$, wherein $x_3 = \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + x_1 + x_2$ and $y_3 = \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_2) + y_1 + c$.
- If $P = (x_1, y_1) \in E(K)$, with $P \neq -P$, $2P = (x_3, y_3)$ where $x_3 = \left(\dfrac{x_1^2 + a}{c}\right)^2$ and $y_3 = \left(\dfrac{x_1^2 + a}{c}\right)(x_1 + x_3) + y_1 + c$.

Group Order. If E is an elliptic curve defined over a field K, the number of points in $E(K)$ is denoted $\#E(K)$. If K is a finite field $GF(q)$, as for any $x \in K$ there are at most two $y \in K$ such that $(x, y) \in E(K)$ (and $\infty \in E(K)$), $\#E(K) \in [1, 2q + 1]$. A tighter bound is given by Hasse's theorem (see [WAS08, chap. 4]).

Theorem 1 (Hasse)**.** *If E is an elliptic curve over $GF(q)$, then $\|q+1-\#E(GF(q))\| \leqslant 2\sqrt{q}$, or equivalently $\#E(GF(q)) = q + 1 - t$, for some t, with $\|t\| \leqslant 2\sqrt{q}$; t is said to be the* trace *of E over $GF(q)$.*

Hasse's theorem provides a bound for $\#E(GF(q))$, but does not provide a method to compute this quantity. Shoof [SCH85] proposes an algorithm which computes $\#E(GF(q))$ in $\mathcal{O}((\log q)^6)$ time complexity. Schoof's algorithm was improved by Elkies and Atkin, the improved variant is now termed as the SEA (Schoof–Elkies–Atkin) algorithm [SCH95]. Satoh [SAT00] proposed an alternative algorithm which is often faster than SEA when $q = p^e$ for a small prime $p \geqslant 3$; subsequent works, among which [FOU00, SKJ03], deal with the usual cryptographic case of $p = 2$.

If E is an elliptic curve defined over a field $GF(q)$ of characteristic p, its order $\#E(GF(q))$ can be used to define supersingularity. Indeed, E is supersingular if p divides its trace t. An elliptic curve E defined over $GF(q)$, is also defined over any extension $GF(q^n)$ of $GF(q)$. The group of the $GF(q^n)$–rational points contains $E(GF(q))$, and if $\#E(GF(q))$ is known, the order of $E(GF(q^n))$ can be efficiently computed using the following.

Theorem 2. *Let E be an elliptic curve defined over $GF(q)$ with order $\#E(GF(q)) = q + 1 - t$. For all $n \geqslant 2$, the order of E over $GF(q^n)$ is $\#E(GF(q^n)) = q^n + 1 - V_n$, where $\{V_{n,n\in\mathbb{N}}\}$ is defined by $V_0 = 2$, $V_1 = t$, and $V_n = V_1 V_{n-1} - qV_{n-2}$ for $n \geqslant 2$.*

2.3 Coordinate Systems and the Group Law

The affine addition formulas require an inversion, which is much slower than a multiplication (see, for instance, [HAN03, table 5.3, p. 220] for efficiency comparisons between inversion and multiplication implementations). In many uses of cryptographic schemes on modern (efficient) computers, this difference is not prohibitive. However, for servers with a large amount of computations, the distinction between different addition formulas efficiency becomes relevant. In this section, we discuss some alternative coordinate systems and addition formulas for non–supersingular elliptic curves over the prime and binary fields (see [HAN03, chap. 3] and [COH05b, chap. 13, 24] for a broader treatment).

2.3.1 Coordinate Systems for Elliptic Curves over Prime Fields

We suppose elliptic curves represented in their simplified Weierstrass forms. Let E be an elliptic curve given by $E:\ y^2 = x^3 + ax + b$. Recall that if $Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ are two rational points, with $Q_1 \neq \pm Q_2$, their sum is $Q_3 = Q_1 + Q_2 = (x_3, y_3)$ where is x_3 and y_3 are as follows,

$$x_3 = \lambda^2 - x_1 - x_2, \tag{2.1}$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \tag{2.2}$$

wherein $\lambda = \dfrac{y_2 - y_1}{x_2 - x_1}$.

And, the double of $Q_1 = (x_1, y_1)$ is $2Q_1 = (x_3, y_3)$ where x_3 and y_3 are given by

$$x_3 = \lambda^2 - 2x_1, \tag{2.3}$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \tag{2.4}$$

where $\lambda = \dfrac{3x_1^2 + a}{2y_1}$.

For affine coordinates, the addition and doubling costs are respectively I + 2M + S and I + 2M+ 2S, where I, M, and S stand respectively for *i*nversion, field *m*ultiplication, and *s*quaring (the field additions are neglected).

Projective Coordinates. The standard projective representation of the curve E is given by $E:\ y^2z = x^3 + axz^2 + bz^3$. As rational points are represented as elements $(x : y : z)$ of the projective space $\mathbb{P}^2(K)$, it is possible to "clear" the denominators in the affine formulas. If $Q_1 = (x_1 : y_1 : z_1)$ and $Q_2 = (x_2 : y_2 : z_2)$ are two rational points, with $Q_1 \neq \pm Q_2$, their sum is $Q_3 = (x_3 : y_3 : z_3)$, where

$$x_3 = uw, \tag{2.5}$$
$$y_3 = u(v^2x_1z_2 - w) - v^3y_1z_2, \tag{2.6}$$
$$z_3 = v^3z_1z_2, \tag{2.7}$$

wherein $u = y_2z_1 - y_1z_2$, $v = x_2z_1 - x_1z_2$, and $w = u^2z_1z_2 - v^3 - 2v^2x_1z_2$.

The double of Q_1 is $Q_3 = 2Q_1 = (x_3 : y_3 : z_3)$, where

$$\begin{aligned} x_3 &= 2uw, && (2.8)\\ y_3 &= t(4v - w) - 8y_1^2u^2, && (2.9)\\ z_3 &= 8u^3, && (2.10) \end{aligned}$$

wherein $t = az_1^2 + 3x_1^2$, $u = y_1z_1$, $v = ux_1y_1$, and $w = t^2 - 8v$.

Using projective coordinates, addition and doubling operations require respectively 12M + 2S and 7M + 5S.

Jacobian Coordinates. A modification of the projective coordinates leads to a faster doubling operation. Let $(x : y : z)$ represent the affine point $(x/z^2, y/z^3)$. Notice that for any positive integers c and d, we have an equivalence relation on the set of non–zero triples over $GF(p)$ by defining (x_1, y_1, z_1) and (x_2, y_2, z_2) equivalent if $x_1 = \lambda^c x_2$, $y_1 = \lambda^d y_2$, and $z_1 = \lambda z_2$ for some $\lambda \in GF(p)$. And, if $c = 2$ and $d = 3$, the projective point $(x : y : z)$, with $z \neq 0$ corresponds to the affine point $(x/z^2, y/z^3)$. The equation of the curve is given by $E : \ y^2 = x^3 + axz^4 + bz^6$. The opposite of $(x : y : z)$ is $(x : -y : z)$, and the point at infinity is $(1 : 1 : 0)$. With this point representation, if $Q_1 = (x_1 : y_1 : z_1)$ and $Q_2 = (x_2 : y_2 : z_2)$, with $Q_1 \neq \pm Q_2$, their sum is $Q_3 = (x_3 : y_3 : z_3)$ where

$$\begin{aligned} x_3 &= -v^3 - 2rv^2 + w^2, && (2.11)\\ y_3 &= -tv^3 + (rv^2 - x_3)w, && (2.12)\\ z_3 &= vz_1z_2, && (2.13) \end{aligned}$$

wherein $r = x_1z_2^2$, $s = x_2z_1^2$, $t = y_1z_2^3$, $u = y_2z_1^3$, $v = s - r$, and $w = u - t$.

And, the double of Q_1, is given by $Q_3 = (x_3 : y_3 : z_3)$ where

$$\begin{aligned} x_3 &= -2v + w^2, && (2.14)\\ y_3 &= -8y_1^4 + (v - x_3)w, && (2.15)\\ z_3 &= 2y_1z_1, && (2.16) \end{aligned}$$

wherein $v = 4x_1y_1^2$, and $w = 3x_1^2 + az^4$.
Using this point representation, addition and doubling now require respectively 12M + 4S and 4M + 6S. Moreover, when $a = -3$, another enhancement is possible, as $w = 3(x_1^2 - z_1^4) = 3(x_1 + z_1^2)(x_1 - z_1^2)$, it can be computed using one squaring and one multiplication. The doubling operation takes 4M + 4S. The NIST curves over prime fields are chosen with $a = -3$ for this reason [IEE00, IEE09].

Even if doubling is enhanced when using Jacobian coordinates, the general points addition remains slower than for projective coordinates. The addition operation is improved, when a point Q is represented as (x, y, z, z^2, z^3). This point representation is termed *Chudnovsky coordinates*; the addition and doubling formulas are the same as for Jacobian coordinates, however, the costs reduce respectively to 11M + 3S and 5M + 6S.

Modified Jacobian coordinates. The modified Jacobian coordinates are mainly the same as Jacobian coordinates, except that a point Q is represented as (x, y, z, az^4) and $t = 8y_1^2$ (the notations are the same as for Jacobian coordinates), so $y_3 = s(r - x_3) - t$ and $az_3^4 = 2t(az_1^4)$. The addition cost is then 13M + 6S, and a doubling operation requires 4M + 4S.

Other Coordinate systems are possible, Edward coordinates [EDW07, BER08] or Jacobi–quartic coordinates [BIL03, DUQ07] for instance, on certain curves. It is also possible to perform additions or doubling operations between points with different representations, and give the result in a third coordinate system; these operations are termed *mixed coordinates*. Many combinations are possible, we only give in Tables 2.1 and 2.2 the operation counts for the most efficient ones; the characters in bold indicate the coordinate systems; the notation $\mathbf{C}_1$+$\mathbf{C}_2 \rightsquigarrow \mathbf{C}_3$, from [HAN03, chap. 3], means the addition of two points given in the $\mathbf{C}_1$ and $\mathbf{C}_2$ coordinate systems, with a result given in the coordinate system $\mathbf{C}_3$.

Table 2.1: Operation counts for point addition and doubling. The **A**, **P**, **J**, **M**, and **C** stand respectively for Affine coordinates, Standard Projective coordinates, Jacobian coordinates, Modified Jacobian coordinates, and Chudnovsky coordinates.

General Addition		**Doubling**	
A	I + 2M+ S	**A**	I + 2M+ 2S
P	12M + 2S	**P**	7M + 5S
J	12M + 4S	**J**	4M + 6S
C	11M + 3S	**C**	5M + 6S
M	13M + 6S	**M**	4M + 4S

Table 2.2: Operation counts for mixed point addition and doubling.

Mixed Addition		**Mixed Doubling**	
C + **C** $\rightsquigarrow$ **J**	10M + 2S	**2M** $\rightsquigarrow$ **C**	4M + 5S
C + **J** $\rightsquigarrow$ **J**	11M + 3S	**2M** $\rightsquigarrow$ **J**	3M + 4S
C + **C** $\rightsquigarrow$ **M**	11M + 4S	**2A** $\rightsquigarrow$ **C**	3M + 5S
J + **C** $\rightsquigarrow$ **M**	12M + 5S	**2A** $\rightsquigarrow$ **M**	3M + 4S
M + **C** $\rightsquigarrow$ **M**	12M + 5S	**2A** $\rightsquigarrow$ **J**	2M + 4S
		J + **A** $\rightsquigarrow$ **M**	9M + 5S
		M + **A** $\rightsquigarrow$ **M**	9M + 5S
		C + **A** $\rightsquigarrow$ **M**	8M + 4S
		C + **A** $\rightsquigarrow$ **C**	8M + 3S
		J + **A** $\rightsquigarrow$ **J**	8M + 3S
		M + **A** $\rightsquigarrow$ **J**	8M + 3S
		A + **A** $\rightsquigarrow$ **M**	5M + 4S
		A + **A** $\rightsquigarrow$ **C**	5M + 3S

2.3.2 Coordinate Systems for Elliptic Curves over Binary Fields

We consider here, the non–supersingular curves $E: y^2+xy = x^3+ax^2+b$. We do not consider here the arithmetic of supersingular curves; these curves come with efficiently computable pairings, which even if constructive in pairing based cryptography (see, for instance, [COH05b, chap. 24]), makes the ECDLP easier to solve than on non–supersingular curves.

Recall that the opposite of a point $Q = (x, y)$ is $-Q = (x, x+y)$, and if $Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$, with $Q_1 \neq \pm Q_2$, their sum is $Q_3 = (x_3, y_3)$ where

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \tag{2.17}$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \tag{2.18}$$

wherein $\lambda = \dfrac{y_1 + y_2}{x_1 + x_2}$.

And, the double of $Q_1 = (x_1, y_1)$ is $Q_3 = (x_3, y_3)$ where

$$x_3 = \lambda^2 + \lambda + a, \tag{2.19}$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \tag{2.20}$$

wherein $\lambda = x_1 + \dfrac{y_1}{x_1}$.

Affine addition and doubling operations have the same cost, namely I + 2M + S; where I, M, and S still refer to *i*nversion, *m*ultiplication and *s*quaring.

Projective Coordinates. In projective coordinates, the curve is given by an equation $E: y^2z + xyz = x^3 + ax^2z + bz^3$; a point $(x : y : z)$, with $z \neq 0$ represents the affine point $(x/z, y/z)$, and the point at infinity is $\infty = (0 : 1 : 0)$. If Q_1, Q_2 are two points given by $Q_1 = (x_1 : y_1 : z_1)$ and $Q_2 = (x_2 : y_2 : z_2)$, with $Q_1 \neq \pm Q_2$, their sum is $(x_3 : y_3 : z_3)$ where

$$x_3 = sv, \tag{2.21}$$
$$y_3 = tz_2(rx_1 + sy_1) + v(r + s), \tag{2.22}$$
$$z_3 = s^3u, \tag{2.23}$$

wherein $r = y_1z_2 + z_1y_2$, $s = x_1z_2 + z_1x_2$, $t = s^2$, $u = z_1z_2$, and $v = u(r^2 + rs + at) + st$.

And, if $Q_1 = (x_1 : y_1 : z_1)$, its double is $(x_3 : y_3 : z_3)$ wherein

$$x_3 = tv, \tag{2.24}$$
$$y_3 = v(s + t) + r^2t, \tag{2.25}$$
$$z_3 = tu, \tag{2.26}$$

where $r = x_1^2$, $s = r + y_1z_1$, $t = x_1z_1$, $u = t^2$, and $v = s^2 + st + au$.

The addition and doubling costs are respectively 16M + 2S and 8M + 4S; and if one of the points is given in affine coordinates, i.e., z_1 or z_2 equals 1, the point addition cost reduces to 12M + 2S.

Jacobian Coordinates. The curve is given here by $E: y^2+xyz = x^3+ax^2z^2+bz^6$; a point $(x : y : z)$, with $z \neq 0$, corresponds to the affine point $(x/z^2, y/z^3)$. The point at infinity is $(1 : 1 : 0)$, and the opposite of a point $(x : y : z)$ is $(x : xz + y : z)$.
If $Q_1 = (x_1 : y_1 : z_1)$ and $Q_2 = (x_2 : y_2 : z_2)$, are two rational points with $Q_1 \neq \pm Q_2$, their sum $Q_1 + Q_2$ is given $(x_3 : y_3 : z_3)$, where

$$\begin{aligned} z_3 &= r'z_2, && (2.27)\\ x_3 &= az_3^2 + wt' + v^3, && (2.28)\\ y_3 &= t'x_3 + r'^2s', && (2.29) \end{aligned}$$

wherein $v = r + s$, $w = t + u$, $r' = vz_1$, $s' = wx_2 + r'y_2$, $z3 = r'z_2$, and $t' = w + z_3$, with $r = x_1z_2^2$, $s = x_2z_1^2$, $t = y_1z_2^3$, and $u = y_2z_1^3$.

The double of $Q_1 = (x_1 : y_1 : z_1)$, is given by $Q_3 = (x_1 : y_1 : z_1)$, where

$$\begin{aligned} x_3 &= s + bt^4, && (2.30)\\ z_3 &= x_1t, && (2.31)\\ y_3 &= sz_3 + x_3(r + y_1z_1 + z_3), && (2.32) \end{aligned}$$

with $r = x_1^2$, $s = r^2$, and $t = z_1^2$.

The addition cost is 16M + 3S, a doubling operation needs 5M + 5S. If one of the inputs is in affine coordinates, the addition cost reduces to 11M + 3S. Notice also that if the coefficient a equals 0 or 1, one multiplication less is needed in the addition.

López–Dahab Coordinates. In this coordinate system the curve is given by an equation $E: y^2 + xyz = x^3z + ax^2z^2 + bz^4$. A point $(x : y : z)$ with $z \neq 0$ represents the affine point $(x/z, y/z^2)$; the point at infinity is $(1 : 0 : 0)$, the opposite of $(x : y : z)$ is $(x : xz + y : z)$.

If $Q_1 = (x_1 : y_1 : z_1)$ and $Q_2 = (x_2 : y_2 : z_2)$ are two points, with $Q_1 \neq \pm Q_2$, their sum is given by $(x_3 : y_3 : z_3)$, where

$$\begin{aligned} x_3 &= r(s' + u) + s(t + r'), && (2.33)\\ z_3 &= wz_1z_2, && (2.34)\\ y_3 &= w(ru' + wr') + x_3(u' + z_3), && (2.35) \end{aligned}$$

wherein $r = x_1z_2$, $s = x_2z_1$, $t = r^2$, $u = s^2$, $v = r + s$, $w = t + u$, $r' = y_1z_2^2$, $s' = y_2z_1^2$, $t' = r' + s'$, and $u' = t'v$.

The double of a point $Q_1 = (x_1, y_1, z_1)$ is (x_3, y_3, z_3) where

$$x_3 = t^2 + s, \tag{2.36}$$
$$z_3 = rt, \tag{2.37}$$
$$y_3 = x_3(y_1^2 + az_3 + s) + z_3 s, \tag{2.38}$$

wherein $r = z_1^2, \quad s = br^2, \quad$ and $t = x_1^2$.

The addition cost in this coordinate system is 13M + 4S, a doubling operation requires 5M + 4S. When one of the points is given in affine coordinates, the addition cost reduces to 9M + 5S. The formulas are the following.

$$x_3 = r^2 + t(r + s^2 + at), \tag{2.39}$$
$$z_3 = t^2, \tag{2.40}$$
$$y_3 = (u + x_3)(rt + z_3) + (y_2 + x_2)z_3^2, \tag{2.41}$$

wherein $r = y_1 + y_2 z_1^2, s = x_1 + x_2 z_1, \quad t = sz_1, \quad$ and $u = x_2 z_3$.

As for the odd characteristic case, mixed coordinates addition and doubling are possible. Many combinations are possible, we only give in Table 2.3 the operation costs for the most efficient ones; notice that even if the coefficient a of a curve may be small, we count a multiplication by a (which may be computed using few additions when a is small) as a full multiplication.

Table 2.3: Operation counts for addition and doubling. The **A**, **P**, **J**, and **LD** stand respectively for Affine, Standard Projective, Jacobian, and López–Dahab.

Doubling		**Addition**	
P	8M + 4S	**P**	16M + 2S
J	5M + 5S	**J**	16M + 3S
LD	5M + 4S	**LD**	13M + 4S
2A $\rightsquigarrow$ **P**	6M + 2S	**P** + **A** $\rightsquigarrow$ **P**	12M + 2S
2A $\rightsquigarrow$ **LD**	3M + 3S	**J** + **A** $\rightsquigarrow$ **J**	11M + 3S
2A $\rightsquigarrow$ **M**	2M+2S	**LD** + **A** $\rightsquigarrow$ **LD**	9M + 5S
A	I + 2M + S	**A** + **A** $\rightsquigarrow$ **LD**	6M + 2S
		A + **A** $\rightsquigarrow$ **J**	5M + S
		A	I + 2M + S

2.4 Scalar Multiplication

This section deals with methods for computing kP, where P is a rational point of order n of an curve E defined over a finite field $GF(q)$, and k is an integer in $[1, n-1]$. We suppose that $\#E(GF(q)) = nh$, with h small, and k represented as a binary string $k = (k_{t-1}, \cdots, k_0)$, where $k_{t-1} \neq 0$ is the most significant bit; we mainly refer to [COH05b, chap 13] in this section.

2.4.1 The Double–and–Add Method

This method is the additive variant of the classical Square–and–multiply exponentiation algorithm; the computation of kP is performed using serial addition or doubling operations depending on the binary representation of k. The Algorithms 2.1 and 2.2 compute kP starting either from the right or left of the binary representation of k.

Algorithm 2.1 Left–to–right binary Double–and–Add method

Input: P, $k = (k_{t-1}, \cdots, k_0)$.
Ouput: $Q = kP$.
(1) Set $Q = \infty$.
(2) **For** j from $t-1$ downto 0 **do**
  (a) $Q = 2Q$.
  (b) If $k_j = 1$ then $Q = Q + P$.
(3) **Return** Q.

Algorithm 2.2 Right–to–left binary Double–and–Add method

Input: P, $k = (k_{t-1}, \cdots, k_0)$.
Ouput: $Q = kP$.
(1) Set $Q = \infty$.
(2) **For** j from 0 to $t-1$ **do**
  (a) If $k_j = 1$ then $Q = Q + P$.
  (b) $P = 2P$.
(3) **Return** Q.

When k is chosen uniformly at random, the expected number of nonzero bits in its representation is $t/2 \approx |q|/2$, as the cofactors are chosen in practice to be small. The Double–and–Add algorithm is expected to require $|q|/2$ point additions and $|q|$ point doubling operations.

2.4.2 Non–Adjacent Forms

If P is a rational point, its opposite can be obtained using few additions in the base field. The cost of a point subtraction is the same as that of an addition. It is then worthwhile to consider point multiplication with representations of k involving signed digits.

A signed–digit representation of an integer k in base β consists of a string $(k_j, \cdots, k_0)$ such that $\|k_i\| < \beta$ and $k = \sum_{i=0}^{j} k_i \beta^i$. The representation is said to be in non–adjacent form if $\beta = 2$ and $k_i k_{i+1} = 0$, for $i = 0, \cdots, j-1$. We denote the NAF of an integer k by NAF(k).

Proposition 1 ([REI60]). *If k is a positive integer, then k has exactly one NAF form.*

The main advantage of the NAF representation is that it has in general fewer non–zero digits than the binary representation. The length of NAF(k) is at most one bit most than that of the binary representation of k; and the average density among all NAFs of

a fixed length l is approximately $l/3$ [MOR90]. Moreover, the NAF of an integer k can be computed by repeatedly dividing k by 2, with a remainder r in $\{-1, 0, 1\}$ chosen so that if k is odd $(k-r)/2$ is event, this ensures the next digit in the remainder to be 0. The procedure is given in Algorithm 2.3 [ARN93][HAN03, chap. 3].

Algorithm 2.3 NAF computation

Input: A positive integer k.
Ouput: NAF(k).
(1) Set $i = 0$.
(2) **While** $k \geqslant 1$ **do**
  (a) If k is odd then $k_i = 2 - (k \bmod 4)$, and $k = k - k_i$.
  (b) Else, $k_i = 0$.
  (c) $k = k/2$ and $i = i + 1$.
(3) **Return** $(k_{i-1}, \cdots, k_0)$.

The Double–and–Add method can be modified to use the NAF form of the scalar k. The procedure is given in Algorithm 2.4. Notice that as the length of NAF(k) is expected to be $|q|/3$, the NAF Double–and–Add method is expected to require $|q|/3$ point additions and $|q|$ doubling operations.

Algorithm 2.4 Binary NAF scalar multiplication

Input: P, k.
Ouput: $Q = kP$.
(1) Compute NAF$(k) = \sum_{i=0}^{l-1} k_1 2^i$.
(2) Set $Q = \infty$
(3) **For** j from $l-1$ downto 0 **do**
  (a) $Q = 2Q$.
  (b) If $k_j = 1$ then $Q = Q + P$.
  (c) If $k_j = -1$ then $Q = Q - P$.
(4) **Return** Q.

A generalization of the NAF scalar multiplication to process a fixed number of digits at a time is also possible [COH05a][HAN03, chap. 3]. If w is an integer greater than 1, then every integer k has a unique representation $k = \sum_{i=0}^{l-1} k_i 2^i$, wherein (1) each k_i is odd or null, (2) $\|k_i\| < 2^{w-1}$ for $i = 0, \cdots, l-1$, and (3) for any w consecutive k_i, at most one among them is non–zero. This expansion is termed width–w NAF, or NAF$_w$ for short, and the NAF$_w$ expansion of an integer k is denoted $(k_{l-1}, \cdots, k_0)_{\text{NAF}_w}$. Avanzi [AVA05] shows that the NAF$_w$ expansion is that of smallest weight among all the expansions with coefficients with absolute values smaller than 2^{w-1}. The average non–zero digits among all NAF$_w$s of length l is approximately $l/(w+1)$. Algorithm 2.5 is a generalization of Algorithm 2.4, it computes NAF$_w(k)$ for $k \in \mathbb{N}^*$.
A generalization of the NAF Double–and–Add algorithm is given in Algorithm 2.6 [HAN03, chap. 3]; the expected running time for the precomputations (the step (2)) is one doubling operation plus $(2^{w-2} - 1)$ additions, the while–loop requires $\dfrac{|q|}{w+1}$ point

Algorithm 2.5 NAF$_w$ computation

Input: A positive integer k.
Ouput: NAF$_w(k)$.
(1) Set $i = 0$.
(2) **While** $k \geqslant 1$ **do**
  (a) If k is odd then $k_i = 2 - (k \bmod 2^w)$, and $k = k - k_i$.
  (b) Else, $k_i = 0$.
  (c) $k = k/2$, and $i = i + 1$.
(3) **Return** $(k_{i-1}, \cdots, k_0)$.

additions and $|q| \approx l$ doubling operations.

Algorithm 2.6 Window NAF scalar multiplication

Input: A width w, k, and P.
Ouput: $Q = kP$.
(1) Compute NAF$_w(k) = \sum_{i=0}^{l-1} k_1 2^i$.
(2) Compute $P_i = iP$ for $i \in \{1, 3, 5, \cdots, 2^{w-1} - 1\}$.
(3) Set $Q = \infty$.
(4) **For** j from $l - 1$ downto 0 **do**
  (a) $Q = 2Q$.
  (b) If $k_j \neq 0$ then
    - If $k_j > 0$ then $Q = Q + P_{k_j}$.
    - Else, $Q = Q - P_{k_j}$.
(5) **Return** Q.

2.4.3 Montgomery Scalar Multiplications

The Montgomery approach introduces an efficient x–coordinate computation; it was proposed for some curves over large characteristic fields [MON87], and later generalized to smaller characteristic curves. An elliptic curve E_M is in Montgomery form if it is given by an equation

$$E_M : \ by^2 = x^3 + ax^2 + x.$$

The arithmetic of Montgomery curves has the following advantage. If $P = (x_1, y_1)$ is a rational point, with $P = (X_1 : Y_1 : Z_1)$ in projective coordinates, and $kP = (X_k : Y_k : Z_k)$, then for all m, n the X and Z–coordinates of $(m + n)P = mP + nP$ can be computed as follows.
If $m \neq n$

$$X_{m+n} = Z_{m-n}((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2, \quad (2.42)$$
$$Z_{m+n} = X_{m-n}((X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n))^2. \quad (2.43)$$

Else,

$$\begin{aligned} 4X_nZ_n &= (X_n+Z_n)^2-(X_n-Z_n)^2, && (2.44)\\ X_{2n} &= (X_n+Z_n)^2(X_n-Z_n)^2, && (2.45)\\ Z_{2n} &= 4X_nZ_n\big((X_n-Z_n)^2+((a+2)/4)(4X_nZ_n)\big). && (2.46) \end{aligned}$$

When $(m-n)P$ is known, and the y–coordinate is not needed, the addition $mP+nP$ requires 4M + 2S, while a doubling requires 3M + 2S. Notice that for many systems (see the ECIES scheme, for instance, in subsection 2.6.1) only the x–coordinates are needed. The y–coordinate $y_n = Y_n/Z_n$ can be recovered using the following formula [OKE01]

$$y_n = \frac{(x_1x_n+1)(x_1+x_n+2a)-2a-(x_1-x_n)^2x_{n+1}}{2by_1}.$$

All Montgomery curves can be transformed into a short Weierstrass curve; however, the converse is not true, as the order of any Montgomery curve is divisible by 4. It is also worthwhile to mention that the elliptic curve cryptography standards (the NIST [NIS03], for instance) recommend the use of curves with cofactors no greater than 4; the curves proposed in standards, are not necessarily transformable into the Montgomery form. Further discussions on the transformability of the Weierstrass curves into the Montgomery form can be found in [OKE01, sect. 4].

A generalization of Montgomery's idea to curves in simplified Weierstrass form was proposed by Brier and Joye [BRI02]. If E is given by

$$E:\ y^2 = x^3+ax+b.$$

The addition formulas are the following (the notations are the same as for the Montgomery form).
If $m \neq n$,

$$\begin{aligned} X_{m+n} &= Z_{m-n}\big(-4bZ_mZ_n(X_mZ_n+X_nZ_m)+(X_mX_n-aZ_mZ_n)^2\big), && (2.47)\\ Z_{m+n} &= X_{m-n}(X_mZ_n-X_nZ_m)^2. && (2.48) \end{aligned}$$

And

$$\begin{aligned} X_{2n} &= (X_n^2-aZ_n^2)^2-8bX_nZ_n^3, && (2.49)\\ Z_{2n} &= 4Z_n\big(X_n(X_n^2+aZ_n^2)+bZ_n^3. && (2.50) \end{aligned}$$

When $(m-n)P$ is known, the addition cost is 9M + 2S, a doubling operation requires 6M +3S; the formula to recover the y–coordinate is

$$y_n = \frac{2b+(x_1x_n+a)(x_1+x_n)-(x_1-x_n)^2x_{n+1}}{2y_1}.$$

López and Dahab [LOP99] propose an extension of Montgomery's idea to the binary case. When E is an non–supersingular curve over a binary field, given by

$$E : y^2 + xy = x^3 + ax^2 + b.$$

The sum $(m+n)P$ (the notations remain the same as for Montgomery's addition), with with $m \neq n$, is given by

$$Z_{m+n} = (X_m Z_n)^2 + (X_n Z_m)^2, \tag{2.51}$$
$$X_{m+n} = Z_{m+n} X_{m-n} + X_m Z_n X_n Z_m. \tag{2.52}$$

And

$$X_{2n} = X_n^4 + bZ_n^4 = (X_n^2 + \sqrt{b} Z_n^2)^2, \tag{2.53}$$
$$Z_{2n} = X_n^2 Z_n^2. \tag{2.54}$$

An addition requires 4M + 1S, the doubling cost is 2M + 3S if $\sqrt{b}$ is precomputed, 2M + 4S otherwise.

With either of the above curve forms, multiplication can be performed as in Algorithm 2.7. At each step of this algorithm, the difference $P_2 - P_1$ equals P, so the Montgomery formulas, can be used for curves in both Montgomery and Weierstrass forms. The computation of kP requires $(6\text{M} + 4\text{S})(|q|/2-1)$ for curves in Montgomery form and simplified Weierstrass curves over binary fields, and $(14\text{M} + 5\text{S})(|q|/2 - 1)$ for simplified Weierstrass curves over prime fields. Notice also that, as at each step a doubling and an addition are performed, this multiplication is also interesting for side–channel attacks [KOC96] resilience.

Algorithm 2.7 Montgomery point multiplication

Input: P, $k = (k_{t-1}, \cdots, k_0)$.
Ouput: $Q = kP$.
(1) Set $P_1 = P$ and $P_2 = 2P$.
(2) **For** j from 0 to $t-1$ **do**
  (a) If $k_j = 0$ then
    $P_1 = 2P_1$ and $P_2 = P_1 + P_2$.
  (b) Else,
    $P_1 = P_1 + P_2$ and $P_2 = 2P_2$.
(3) **Return** P_1.

Depending on the implementation context, other enhancements are possible on point multiplication efficiency. If the point P is a fixed one for instance, precomputation based windowing multiplications are possible [HAN03, chap. 3]. Using the additive variant of Shamir's multiple exponentiation technique [MEN96, Algorithm 14.88], the cost of the computations of the form $k_1P + \cdots + k_jP$, can also be reduced to be roughly equivalent to one point multiplication and half [HAN03, chap. 3].

2.5 The Elliptic Curve Discrete Logarithm (and related) Problem(s)

Loosely speaking, modern cryptography deals with the construction of schemes which are easy to operate but hard to foil. Indeed, almost all of modern cryptography rises or falls with the question of whether or not one–way functions exist. One–way functions are easy to evaluate but hard (on the average) to invert. The elliptic curve discrete logarithm problem, is widely believed to be a one–way function.

Definition 2 (ECDLP)**.** Let E be an elliptic curve over a finite field $GF(q)$, and $P \in E(GF(q))$ a point of order n. The elliptic curve discrete logarithm problem (ECDLP) is: given $Q \in \langle P \rangle$, P, and n, find $l \in [1, n-1]$ such that $lP = Q$. The integer l is said to be the discrete logarithm of Q in base P, and is noted $l = \log_P Q$.

The hardness of the ECDLP is a prerequisite for the security all elliptic curve cryptographic schemes. The best known general purpose method to solve the ECDLP is the combination of the Pohlig–Hellman algorithm [HAN03, chap. 4] and Pollard's rho algorithm [HAN03, TES01a] (see also 2.5.1) which has a running time of $\mathcal{O}(\sqrt{n_1})$ where n_1 is n's the largest prime factor. For (well chosen) elliptic curve parameters such that n is divisible by a sufficiently large prime, to make $\mathcal{O}(\sqrt{n_1})$ operations infeasible ($|n_1| \geqslant 163$ is today sufficient), solving the ECDLP is believed to be infeasible.

For some cryptographic schemes the hardness of the following (EC)DLP related problems is required.

Definition 3 (ECDHP)**.** Let E be an elliptic curve over $GF(q)$, and $P \in E(GF(q))$, $Q_1 = l_1 P, Q_2 = l_2 P \in \langle P \rangle$. The computational Elliptic Curve Diffie–Hellman Problem (ECDHP) is, given P, Q_1, Q_2, and n, find $Q_3 = l_1 l_2 P$.

It is not difficult to see that any efficient ECDLP solver yields an efficient ECDHP solver. The ECDHP is not harder than the ECDLP. However, it is not known whether the converse is true (i.e., whether the ECDHP is as hard as the ECDLP). Given an efficient ECDHP solver there is no know algorithm which efficiently solves the ECDLP, unless in some specific cases. When $\varphi(n)$ (where φ is the Euler totient function) has no large prime factor, Den Boer [DEN88] shows that the ECDLP and ECDHP problems are equivalent. Boneh and Shparlinski [BON01] show that for an elliptic curve E defined over a prime field $GF(p)$, and $P \in E(GF(p))$ of prime order, with the ECDHP difficult in $\langle P \rangle$, no efficient algorithm can predict the least significant bit of the x–coordinate (or the y–coordinate) of the Diffie–Hellman secret point $l_1 l_2 P$ for most elliptic curves isomorphic to E. (This provides some "evidence" that computing the least significant bit of the x–coordinate of of $l_1 l_2 P$ from P, $l_1 P$, and $l_2 P$ is as hard as computing $l_1 l_2 P$.)

For many schemes, the hardness of the ECDHP is not known to be sufficient; loosely speaking, it is required that given P, $l_1 P$, $l_2 P$, and n, no efficient algorithm can learn any information about $l_1 l_2 P$. This is formalized through the Elliptic Curve Decisional Diffie–Hellman Problem (ECDDHP).

Definition 4 (ECDDHP)**.** Let E be an elliptic curve over a finite field $GF(q)$, $P \in E(GF(q))$ a point of order n. The Elliptic Curve Decisional Diffie–Hellman Problem

(ECDDHP) is: given P, l_1P, l_2P, n, and $W = l_3P$, determine whether or not W equals l_1l_2P.

Any efficient algorithm which solves the ECDHP yields an efficient ECDDHP solver. The ECDDHP is not harder than the ECDHP; it is not known whether or not the converse is true.

2.5.1 Attacks on the ECDLP

Even widely believed, there is no proof that the ECDLP is intractable. (Notice that a proof of the non–existence of a polynomial–time algorithm for the ECDLP would imply that the complexity class **P** is different from **NP**.)

The Pohlig–Hellman Attack. The Pohlig–Hellman approach [POH78] reduces the computation of $l = \log_P Q$ to computations of discrete logarithms in prime order subgroups of $\langle P \rangle$. Suppose the prime factorization of n known, $n = \prod p_i^{e_i}$; the idea is to find $l \bmod p_i^{e_i}$ for each i, and use the Chinese Remainder Theorem (CRT) to obtain $l \bmod n$. Using this idea, the computation of $l \bmod n$ reduces to computations of $l \bmod p^e$, with p prime. Suppose that l writes in base p as $l = z_0 + z_1 p + z_2 p^2 + \cdots$, with $0 \leqslant z_i \leqslant p-1$; $l \bmod p_i^e$ is computed by successively determining $z_0, z_1, \cdots, z_{e-1}$. The computations are the following:

(1) Compute $\mathcal{T} = \left\{ j\frac{n}{p}P, \text{ for } 0 \leqslant j \leqslant p-1 \right\}$.

(2) Compute $Q_0 = \frac{n}{p}Q$, this will equal an element $z_0\frac{n}{p}P$ of $\mathcal{T}$, and stop if $e = 1$.

(3) Compute $Q_1 = Q - z_0P$ and $\frac{n}{p^2}Q_1$, this will equal an element $z_1\frac{n}{p}P$ of $\mathcal{T}$, and stop if $e = 2$.

(4) More generally, if $z_0, \cdots, z_{r-1}$ and $Q_1, \cdots, Q_{r-1}$ are already computed, to compute z_r, one does the following:
- Compute $Q_r = Q_{r-1} - z_{r-1}p^{r-1}P$.
- Determine z_r such that $\frac{n}{p^{r+1}}Q_r = z_r\frac{n}{p}P$.

(5) Compute $l \bmod p^e = z_0 + z_1 p + z_2 p^2 + \cdots + z_{e-1}p^{e-1}$.

As $\frac{n}{p}P = P_0$, has order p, we have $Q_0 = \frac{n}{p}Q = \frac{n}{p}(z_0 + z_1 p + \cdots)P = z_0\frac{n}{p}P$; hence $z_0 = \log_{P_0} Q_0$. Next, $\frac{n}{p^2}Q_1 = \frac{n}{p^2}(l - z_0)P = \frac{n}{p^2}(z_0 + z_1 p + z_2 p^2 + \cdots - z_0)P = z_1\frac{n}{p}P = z_1P_0$. Similarly the computations yield $z_2, z_3, \cdots, z_{e_1}$; besides, it is not needed to continue, as $l \bmod p^e$ is known. The ECDLP in $\langle P \rangle$ is not harder that in its prime order subgroups.

Shanks' Baby Step Giant Step Attack. Shanks' method [SHA71, TES01b] uses a time–memory trade–off; it requires approximately $\sqrt{n}$ operations and $\sqrt{n}$ space complexity. The approach, which is rather simple, is the following.

(1) Compute $m = \lceil \sqrt{n} \rceil$ and mP.

(2) Compute and store iP for $1 \leqslant i \leqslant m$.

(3) Compute $R_j = Q - j(mP)$ for $0 \leqslant j < m$, until R_j matches an element of the stored list.
(4) Return $l = i + jm$.

The method is general–purpose and deterministic; however it requires a large storage, which makes the "equivalent" probabilistic methods often preferable.

Pollard's rho Method. The leading idea in the rho method [POL78] is to find two distinct couples (c_1, d_1), (c_2, d_2) such that $c_1P + d_1Q = c_2P + d_2Q$; and compute $l = \log_P Q = (d_2 - d_1)^{-1}(c_1 - c_2) \bmod n$. For this purpose, an iterating function $f : \langle P\rangle \longrightarrow \langle P\rangle$ that approximates a random function is used. Since $\langle P\rangle$ is finite, a sequence $(R_i)_{i\in\mathbb{N}}$ with $R_0 \in_R \langle P\rangle$ and $R_{i,i>0} = f(R_{i-1})$ will eventually collide. Thus there exists some λ and μ, called respectively *period* and *preperiod*, such that $R_0, \cdots, R_{\lambda+\mu-1}$ are pairwise distinct and $R_{\lambda+\mu} = R_\mu$. If f is supposed to be random, the expected values of the period and preperiod are $\lambda \approx \sqrt{\pi n/8}$ and $\mu \approx \sqrt{\pi n/8}$. A convenient way to build such iterating functions [POL78, TES01a] is to take a (pseudo)random partition $\{\mathcal{P}_1, \cdots, \mathcal{P}_L\}$ of $\langle P\rangle$ (with subsets having roughly the same size) and $\gamma_j, \delta_j \in_R [0, n-1]$ for each $\mathcal{P}_j$ (L is the number of branches); then f can be defined $\langle P\rangle \ni R \overset{f}{\longmapsto} R + \gamma_j P + \delta_j Q$ if $R \in \mathcal{P}_j$. Collision search is performed using *Floyd's cycle finding algorithm* [KNU81, Exercise 3.1–6] in which one starts with (R_1, R_2) and compute $(R_i, R_{2i})_{i>1}$ until $R_i = R_{2i}$. The required storage for this approach is thus negligible. The expected number of couples that have to be computed until collision is about $1.0308\sqrt{n}$; the method is given in Algorithm 2.8.

Algorithm 2.8 Pollard's rho algorithm

Input: P, n , $Q \in \langle P\rangle$.
Ouput: $l = \log_P Q$ or "failure".
(1) Choose a partition function $g : \langle P\rangle \longrightarrow \{1, \cdots, L\}$ ($g(R) = j$ if $R \in \mathcal{P}_j$).
(2) **For** j from 1 to L **do**
  (a) choose γ_j, $\delta_j \in_R [0, n-1]$;
  (b) compute $R^{(j)} = \gamma_j P + \delta_j Q$.
(3) Choose $c_1, d_1 \in_R [0, n-1]$ and compute $R_1 = c_1P + d_1Q$.
(4) Set $R_2 = R_1$, $c_2 = c_1$, and $d_2 = d_1$.
(5) **Repeat**
  (a) $j = g(R_1)$, $R_1 = R_1 + R^{(j)}$, $c_1 = c_1 + \gamma_j \bmod n$, and $d_1 = d_1 + \delta_j \bmod n$;
  (b) **For** i from 1 to 2 **do**
    $j = g(R_2)$, $R_2 = R_2 + R^{(j)}$, $c_2 = c_2 + \gamma_j \bmod n$, and $d_2 = d_2 + \delta_j \bmod n$.
  until $R_2 = R_1$.
(6) **If** $d_1 = d_2$ **return** "failure".
(7) **Return** $l = (c_1 - c_2)(d_2 - d_1)^{-1} \bmod n$.

In a naive parallel implementation of the rho algorithm, with an instance running on each processor until one succeeds, the expected computational effort of each processor before one succeeds is $3\sqrt{\frac{n}{m}}$, when m processors are available [VAN99].

Van Oorschot and Wiener [VAN99] propose a client–server parallelization approach which, when m processors are available, yields a factor m speedup. In this approach, each client processor randomly chooses its own starting point R_{0_j}, but the iterating function is the same for all clients. An easily testable subset of $\langle P \rangle$ is used as a distinguished set; the set of distinguished points can be, for instance, the set of points with the leading t bits of their x–coordinate being zero. When a client processor finds a distinguished point, the point is transmitted to the server which stores it in a sorted list. When the server receives the same distinguished point for the second time, it computes the desired solution. For each client processor, the expected number of iterations before a collision is $\frac{\sqrt{\pi n/2}}{m}$. Let ϑ be the proportion if distinguished points, the expected number of elliptic curve operations per client processor before a collision is found is $\frac{1}{m}\sqrt{\frac{\pi n}{2}} + \frac{1}{\vartheta}$. The overall messages received by the server is $\vartheta\sqrt{2\pi n}$.

Algorithm 2.9 Parallelized Pollard's rho algorithm

Input: P, n , $Q \in \langle P \rangle$.
Ouput: $l = \log_P Q$ or "failure".
(1) Choose a partition function $g : \langle P \rangle \longrightarrow \{1, \cdots, L\}$ ($g(R) = j$ if $R \in \mathcal{P}_j$).
(2) Choose an easily computable distinguishing property for points $\in \langle P \rangle$.
(3) **For** j from 1 to L **do**
  (a) Choose γ_j, $\delta_j \in_R [0, n-1]$.
  (b) Compute $R^{(j)} = \gamma_j P + \delta_j Q$.
(4) Each of the client processors does the following:
  (a) Choose $c, d \in_R [0, n-1]$ and compute $R = cP + dQ$.
  (b) **Repeat**
    (i) If R is distinguished point, send (c, d, R) to the server.
    (ii) Compute $j = g(R)$.
    (iii) Compute $R = R + R^{(j)}$, $c = c + \gamma_j \bmod n$, and $d = d + \delta_j \bmod n$.
  until the server receives the same distinguished point twice
(5) Let (c_1, d_1, R_d) and (c_2, d_2, R_d) be the two triples associated with the distinguished point R_d received twice.
(6) **If** $d_1 = d_2$ **return** "failure".
(7) **Return** $l = (c_1 - c_2)(d_2 - d_1)^{-1} \bmod n$.

Isomorphism Attacks. Suppose that the order n of $\langle P \rangle$ is prime, and let $\mathcal{G}$ be a group of order n. Both $\langle P \rangle$ and $\mathcal{G}$ are cyclic of order n, hence isomorphic. If the isomorphism is efficiently computable, the ECDLP in $\langle P \rangle$ can be reduced to a DLP in $\mathcal{G}$.

If E defined over $GF(p)$, is an anomalous curve (i.e., $\#E(GF(p)) = p$) the group of rational points $E(GF(p))$ isomorphic to the additive group of $GF(p)$. As simultaneously shown in [SAT98, SEM98, SMA99] the isomorphism between $E(GF(p))$ and the additive group of $GF(p)$ can be efficiently computed. The ECDLP in prime–field anomalous curves reduces to an additive DLP problem in the additive group of $GF(p)$,

which can be efficiently solved.

If in addition to being prime, the order n of $E(GF(q))$ satisfies $\gcd(n, q) = 1$; let k be the smallest integer satisfying $q^k = 1 \bmod n$. As k, said to be the *embedding degree*, is the order of q modulo n, it divides $n - 1$. And, as n divides $q^k - 1$, the multiplicative group $GF(q^k)^\times$ has a unique subgroup $\mathcal{G}$ of order n. The MOV pairing attack [MEN93], builds an isomorphism between $\langle P \rangle$ and $\mathcal{G}$ when n does not divide $q - 1$, while the Frey–Rück attack [FRE94] builds an isomorphism between $\langle P \rangle$ and $\mathcal{G}$ without requiring this condition.

For non–supersingular elliptic curves over binary fields, Frey and Gangl [FRE98] propose the idea of using the Weil descent [BLA00, chap. 8][COH05b, chap. 7, 22], also termed scalar restriction, to reduce the ECDLP in $E(GF(2^m))$ to a discrete logarithm problem in the jacobian of a hyperelliptic curve of larger genus over a proper subfield $GF(2^l)$ of $GF(2^m)$. Gaudry, Hess, and Smart (GHS) [GAU02] give an efficient algorithm which reduces the ECDLP to the discrete logarithm problem in a Jacobian of a hyperelliptic curve over a proper subfield $GF(2^l)$ of $GF(2^m)$. Since subexponential running–time algorithms are known for the discrete logarithm problem in higher genus curves [COH05b, chap. 20, 21], this yields a possible method of attack against the ECDLP.

Tuned Implementation Outcomes

Despite the parallel variant and possible optimizations in Pollard's rho approach (the use of equivalence classes [WIE99] or Teske's r–addings [TES01a]), the published effective records for the elliptic curve discrete logarithm problem still remain somewhat moderate. The most recent record solved an instance of the ECDLP over a 112–bit prime field, the SEC 2 'secp112r1' standard curve [BOS09]; it required 62.6 PlayStation 3 (PS3) years. The computations, sometimes interrupted, took 7 months of calender time, and required 0.6 Terabyte of disk space. According to [BOS09], a continuous execution of their code on a cluster of more than 200 PS3 would take 3.5 months. (Notice that they do not indicate the exact number of PS3s they used.) The previous records, dated respectively from 2004 and 2002, were on Certicom challenges ECC2–109 and ECCp–109 [CERT09]. The record on ECC2–109 and ECCp–109 required respectively 17 and 18 months of calender time [CERT09].

Surprisingly, even if Teske's r–addings [TES01a] are used for the records, the negation map which theoretically yields a factor $\sqrt{2}$ speed–up, was not used for any of the records. In fact, the negation map yields fruitless cycles [BOS10], and it does not effectively achieve its theoretical speed–up; its use requires further fruitless cycle handling techniques. A discussion on the effective use of the negation map can be found in [BOS10].

Security Precautions for Cryptographic Curves

As a consequence of the aforementioned attacks, a curve E defined over $GF(q)$ is cryptographically interesting, if $\#E(GF(q))$ is divisible by a large prime n. Having $|n| \geqslant 163$ is sufficient for resistance against the Pohlig–Hellman, Pollard's rho, and

Shanks' BSGS attacks; for an optimal resistance to these attacks E can be chosen such that $\#E(GF(q)) = nh$ with $h \leqslant 4$. To avoid the prime field anomalous curves, $\#E(GF(q))$ should be different from q. The Weil and Tate pairing attacks are infeasible, if n does not divide $q^k - 1$ for $1 \leqslant k \leqslant t$ with t large enough ($t \geqslant 20$ is sufficient). Menezes and Qu [MEN01] show that the GHS attack fails for all cryptographically interesting elliptic curves over $GF(2^m)$ with m prime and in $[160, 600]$. To guard against attacks that may be discovered in the future, one may use curves chosen at random, as long as the mentioned security precautions are satisfied.

2.6 Basic Elliptic Curves Based Schemes

The basic public key security services can be built using elliptic curves; in this section we recall some basic elliptic curve based schemes. In practice, (well–chosen) curves are precomputed and shared between a group of parties. In the continuation, we suppose that the considered parties choose their keys in a *public domain parameters*. (Notice that all the protocols described in the next chapters in a generic group, can be used with elliptic curve groups.)

Definition 5 (Domain parameters)**.** A domain parameters $\Psi = (q, FR, S, a, b, P, n, h)$ consists of:

(a) A field order q.
(b) An indication of the representation of the elements of $GF(q)$.
(c) For randomly generated curves, a seed S used to generated verifiably at random the coefficients a and b.
(d) The coefficients $a, b \in GF(q)$ defining the curve (i.e., $y^2 = x^3 + ax + b$ if $GF(q)$ is a prime field or an optimal extension field[1], and $y^2 + xy = x^3 + ax^2 + b$ if $GF(q)$ is a binary field).
(e) A point $P = (x, y) \in E(GF(q))$ (represented in affine coordinates) of prime order; P is said to be the *base point.*
(f) The integer n is the order of P, and $hn = \#E(GF(q))$; h is said to be the *cofactor.*

Domain parameters must be chosen to avoid the Pohlig–Hellman, Pollard's rho, and isomorphism attacks. Given a valid domain parameters, a *key pair generation* consists in:

- Choosing $d \in_R [1, n-1]$.
- Computing $Q = dP$.
- The public key is Q, the private part is d.

The public key Q should be available to any party, which may communicate with its owner; in addition, the identity of the key owner must be associated with the key in a way which is verifiable by all parties. Certification Authorities (CAs) are used to generate certificates attesting this association.

[1] An optimal extension field is a field $GF(p^m)$ such that (1) $p = 2^n - c$ for some integers n, c with $\log_2 |c| \leqslant n/2$, and (2) an irreducible polynomial $f(z) = z^m - w$ exists in $GF(p)[z]$; efficient arithmetics can be performed on these fields [BAIL98, BAIL01].

2.6.1 The Elliptic Curve Integrated Encryption Scheme

The ECIES scheme is an elliptic curve variant of the famous ElGamal public key encryption scheme [STI95, chap. 6]. It was proposed by Bellare and Rogaway [BEL97]; Cramer and Shoup [CRA04] showed the scheme secure against adaptive chosen ciphertext attacks, under the Random Oracle model and the elliptic curve Gap Diffie–Hellman assumption (which is: given an efficient ECDDHP solver, the ECDHP problem remains hard). The ECIES scheme is standardized in ANSI X9.63 [ANS01b] and IEEE P1363 [IEE00]. An ECIES encryption is as follows; KDF is a key derivation function, Enc is a symmetric encryption scheme, and MAC a message authentication scheme.

Algorithm 2.10 ECIES Encryption

Input: A domain parameters $\Psi = (q, FR, S, a, b, P, n, h)$, a public key Q, and a message m.
Output: A ciphertext $c = (R, C, t)$.

(1) Choose $k \in_R [1, n-1]$.
(2) Compute $R = kP$ and $Z = hkQ$.
- If $Z = \infty$, go to step (1).
- Else, destroy k.

(3) Compute $(K_1, K_2) = KDF(x_Z, R)$, where x_Z is the x–coordinate of Z.
(4) Compute $C = Enc_{K_1}(m)$, and $t = MAC_{K_2}(C)$.
(5) Return $c = (R, C, t)$.

The design of ECIES is quite simple, the sender provides the receiver with $R = kP$, together with $Z = hkQ = hk(dP)$, where Q and d are the receiver's public and private keys. (The use of hkQ instead of kQ guarantees that Z does not belong to the small subgroup of $E(GF(q))$.) The receiver can compute $Z = hdR$, K_1, and K_2; it then authenticates and decrypts the ciphertext c to obtain m. Recall that validating a public key R consists in: (1) verifying that $R \neq \infty$, and x_Q and y_Q are properly represented in $GF(q)$, and (2) verifying that R satisfies the curve defined by a and b, and that $nR = \infty$. The decryption operation is given in Algorithm 2.11.

Algorithm 2.11 ECIES Decryption

Input: A domain parameters $\Psi = (q, FR, S, a, b, P, n, h)$, a private key d, and a ciphertext $c = (R, C, t)$.
Output: A plaintext m or "failure" (i.e, ciphertext rejection).

(1) Validate the public key R, if the validation fails, return "failure".
(2) Compute $Z = hdR$, if $Z = \infty$, return "failure".
(3) Compute $K_1, K_2 = KDF(x_Z, R)$.
(4) Verify that $t = MAC_{K_2}(C)$, if not return "failure".
(5) Return $m = Dec_{K_1}(m)$.

For a honest sender, it is not difficult to see that the decryption algorithm yields the message m.

2.6.2 The Elliptic Curve Digital Signature Algorithm

ECDSA is an elliptic curve analog of the DSA scheme [MEN96, chap. 3]; it seems to appear for the first time in [VAN92] as a response to a NIST request for comments. It is today widely standardized [ANS05, IEE00, FIP00]. The signature generation is as follows; H is $|n|$–bit hash function.

Algorithm 2.12 ECDSA Signature Generation

Input: A domain parameters $\Psi = (q, FR, S, a, b, P, n, h)$, a private key d, and a message m.
Output: A signature $sig = (r, s)$.
(1) Choose $k \in_R [1, n-1]$.
(2) Compute $R = kP$ and convert x_R (the x–coordinate of R) to an integer $\bar{x}_R$.
(3) Compute $r = \bar{x}_R \bmod n$, if $r = 0$, go to step (1).
(4) Compute $e = H(m)$.
(5) Compute $s = k^{-1}(e + dr) \bmod n$; if $s = 0$, go to step (1).
(6) Return $sig = (r, s)$.

The ECDSA signature scheme is proven GMR[2]–secure (i.e., existentially unforgeable against an efficient adaptive chosen message attacker) in the generic group model [BRO05]. But, as shown by Dent [DEN02] security arguments in the generic group model does not necessarily provide assurance in practice; namely [DEN02] describes a signature scheme which is provably secure in the generic group model, but insecure in any specific group.

A signature verification is as in Algorithm 2.13. For a signature $sig = (r, s)$ on a message m; since $s = k^{-1}(e + dr) \bmod n$, it follows that $k = s^{-1}(e + dr) = s^{-1}e + s^{-1}rd = u_1 + u_2 d \bmod n$. Hence the required equality between $\bar{x}_R$ and $\bar{x}_{u_1 P + u_2 Q}$ holds.

Algorithm 2.13 ECDSA Signature Verification

Input: A domain parameters $\Psi = (q, FR, S, a, b, P, n, h)$, a public key Q, and a signature $sig = (r, s)$.
Output: "valid" or "invalid".
(1) Verify that r and s belong to $[1, n-1]$; if not return "invalid".
(2) Compute $e = H(m)$.
(3) Compute $w = s^{-1} \bmod n$.
(4) Compute $u_1 = ew \bmod n$, $u_2 = rw \bmod n$.
(5) Compute $R = u_1 P + u_2 Q$.
(6) If $R = \infty$, return "invalid".
(7) Convert x_R to an integer $\bar{x}_R$, and compute $v = \bar{x}_R \bmod n$.
(8) If $v = r$ return "valid"; else, return "invalid".

[2] Goldwasser, Micali, and Rivest security definition [GOL88]

2.6.3 The Password Authenticated Connection Establishment

The PACE protocol was proposed by the German Federal Office for Information Security (BSI) [BSI10]. The protocol establishes a secure channel between two parties (a chip and a terminal), based only on a weak password. Recall that a key refers to a string with sufficiently large entropy to be resistant to guessing attacks, while a password refers to a short string which may be easily memorized by a human user.

The PACE protocol was designed with travel document systems in mind. Broadly, the protocol divides into four main steps; in the first, the initiator (the chip) provides the responder (the terminal) with a random nonce s encrypted with a key derived from the password. Second, both parties run an ephemeral base point generation protocol. Third, they run an anonymous Diffie–Hellman protocol based on a domain parameters provided by the initiator, and finally derive the shared key. The protocol description is given in Protocol 2.14, wherein KDF_π, KDF_1 and KDF_2 are key derivation functions. The domain parameters validation, may simply consist in verifying that it was signed by a trusted third party; if any verification fails, the run fails and terminates.

PACE is rather a framework, allowing different instantiations of the ephemeral base point generation. In our description, we used the most prominent base point generation protocol; other instantiations are possible [BSI10, BEN09].

The protocol is shown secure in the Abdalla, Fouque and Pointcheval model [ABD05] under a variant of the (EC)DHP problem (the PACE–DH problem), the random oracle model, and the ideal cipher model (which are now known to be equivalent [COR08]). It is not known whether the ECDHP is equivalent to the ECPACE–DHP; however, in generic groups, the two problems are shown to be equivalent [BEN09].

As the security of the PACE protocol relies only on a password π an adversary can guess the right π with probability at least $1/2^{|\pi|}$ and then impersonate one party to another. Loosely speaking, the security arguments show that for any attacker performing $C(l)$ on–line attacks, where $l = 1/2 \log_2 n$, the probability it succeeds is smaller than $C(l)/2^{|\pi|} + \varepsilon(l)$, where $\varepsilon(\cdot)$ is negligible (i.e., for all $c > 0$ there is some k_c such that, $\|\varepsilon(k)\| < k^{-c}$ for all $k > k_c$). In particular, the adversary should not be able to compute the password of any party through an *off–line* dictionary attack by successfully matching password candidates to executions afterwards.

Notice also that, despite of its formal security arguments [BEN09], the PACE protocol is particularly sensitive to ephemeral secret information leakages. The protocol is not only vulnerable to ephemeral DH exponent leakage, but also to a leakage of the ephemeral base point or the final DH secret σ. As one can see, if an attacker can access an ephemeral base point $\tilde{P}$ at an honest party, say $\hat{B}$, it can substitute $\hat{A}$'s ephemeral public key X_A (at step IIIb) with $2P$, for instance, in $\hat{A}$ message to $\hat{B}$, and using $\tilde{P} = (sG + 2X_B)$ and c (which is sent in $\hat{A}$'s first message), recover the password π using an *off–line* exhaustive search.

Protocol 2.14 The PACE Protocol

PROTOCOL MESSAGES:

$\underline{\hat{A}} : \pi, \Psi$		$\underline{\hat{B}} : \pi$
$K_\pi = KDF_\pi(\pi)$,		
$s \in_R [1, n-1]$,		
$c = Enc_{K_\pi}(s)$		
	$\boxed{c, \Psi} \longrightarrow$	
		$K_\pi = KDF_\pi(\pi)$,
		$s = Dec_{K_\pi}(c)$,
		$x_B \in_R [1, n-1]$,
	$\longleftarrow \boxed{X_B = x_B P}$	
$x_A \in_R [1, n-1]$,		
	$\boxed{X_A = x_A P} \longrightarrow$	
		$\tilde{P} = sP + x_B X_A$,
		$y_B \in_R [1, n-1]$,
	$\longleftarrow \boxed{Y_B = x_B \tilde{P}}$	
$\tilde{P} = sP + x_B X_A$,		
$y_A \in_R [1, n-1]$,		
	$\boxed{Y_A = y_A \tilde{P}} \longrightarrow$	
$\sigma_A = y_A Y_B$		$\sigma_B = y_B Y_A$,
$K_{Enc} = KDF_1(\sigma_A)$,		$K_{Enc} = KDF_1(\sigma_B)$,
$K_{Mac} = KDF_2(\sigma_A)$,		$K_{Mac} = KDF_2(\sigma_B)$,
	$\longleftarrow \boxed{t_B = MAC_{K_{Mac}}(Y_B, \Psi)}$	
	$\boxed{t_A = MAC_{K_{Mac}}(Y_A, \Psi)} \longrightarrow$	

I) The initiator $\hat{A}$ does the following:
   (a) Choose $s \in_R [1, n-1]$, compute $K_\pi = KDF_\pi(\pi)$, and $c = Enc_{K_\pi}(s)$.
   (b) Send c and $\Psi = (q, FR, S, a, b, P, n, h)$ to $\hat{B}$.

II) $\hat{B}$ does the following:
   (a) Compute $K_\pi = KDF_\pi(\pi)$, and $s = Dec_{K_\pi}(c)$.
   (b) Validate the domain parameters Ψ.
   (c) Choose $x_B \in_R [1, n-1]$, and send $X_B = x_B P$ to $\hat{A}$.

III) $\hat{A}$ does the following:
   (a) Verify that $X_B \in \mathcal{G}^*$.
   (b) Choose $x_A \in_R [1, n-1]$, and send $X_A = x_A P$ to $\hat{B}$.

IV) $\hat{B}$ does the following:
   (a) Verify that $X_A \in \mathcal{G}^*$.
   (b) Choose $y_B \in_R [1, n-1]$, and send $Y_B = y_B(sP + x_B X_A)$ to $\hat{A}$.

V) $\hat{A}$ does the following:
   (a) Verify that $Y_B \in \mathcal{G}^*$.
   (b) Choose $y_A \in_R [1, n-1]$, and send $Y_A = y_A(sP + x_A X_B)$ to $\hat{B}$.
   (c) Compute $\sigma_A = y_A Y_B$, $K_{Enc} = KDF_1(\sigma_A)$, and $K_{Mac} = KDF_2(\sigma_A)$.

VI) $\hat{B}$ does the following:
   (a) Verify that $Y_A \in \mathcal{G}^*$.
   (b) Compute $\sigma_B = y_b Y_A$, $K_{Enc} = KDF_1(\sigma_B)$, and $K_{Mac} = KDF_2(\sigma_B)$.
   (c) Send $t_B = MAC_{K_{Mac}}(Y_B, \Psi)$ to $\hat{A}$.

VII) $\hat{A}$ verifies that $t_B = MAC_{K_{Mac}}(Y_B, \Psi)$, and sends $t_A = MAC_{K_{Mac}}(Y_A, \Psi)$ to $\hat{A}$.

VIII) $\hat{B}$ verifies that $t_A = MAC_{K_{Mac}}(Y_A, \Psi)$.

IX) The shared session keys are K_{Enc} and K_{Mac}.

2.7 Advantages of Elliptic Curves based Cryptography

Different criteria can be considered when comparing public key scheme families. In practice, principal criteria seem to be functionality, security, and efficiency. The RSA, discrete logarithm (over finite fields), and elliptic curve discrete logarithm families provide the basic functionalities in public key cryptography (encryption, signature, key exchange and distribution); and the hardness of the underlying mathematical problems, which is necessary for the security of the schemes, is well–studied and seems widely believed.

Public key schemes are often used in combination with symmetric schemes. For the same security, against the best known attacks, the elliptic curve parameters can be chosen much smaller than the RSA or finite field discrete logarithm ones. For instance, a 160–bit elliptic curve group order is expected to yield the same security level as a 1024–bit RSA modulus or multiplicative discrete logarithm group order. The differences become particularly important when the desired security level increases; a (well–chosen) 512–bit domain parameter is currently equivalent in security to a 15360–bit RSA modulus. Table 2.4 summarizes the approximate sizes for security equivalence between, symmetric schemes, elliptic curves, RSA, and discrete logarithm based schemes.

Table 2.4: Key sizes (in bits) for equivalent security levels [LAW03, chap. 1].

Symmetric	ECC	RSA/DL
80	160	1024
112	224	2048
128	256	3072
192	384	8192
256	512	15360

The advantages of using significantly smaller parameters in elliptic curve cryptography include efficiency and storage reduction (faster computations and smaller keys). Even if, for small public exponents ($e = 65537$, for instance), public key operations for RSA schemes can be expected faster than for elliptic curve schemes, the other operations (signature generation, decryption, etc.) are faster for elliptic curves than for RSA or finite field discrete logarithm based schemes. The advantage of elliptic curve schemes can be particularly important in computationally limited processing environments with limited storage or bandwidth.

2.8 Elliptic Curve Cryptography Standards Activities

Elliptic curve cryptography is now widely used in industry, and cryptographic standard bodies often provide standards dealing with elliptic curve cryptography. In this subsection, we recall some of these bodies and the schemes they standardize, our survey is not exhaustive; indeed an exhaustive survey would be difficult, if not impossible, as almost every country defines its own cryptographic standards, ANSSI "Agence Nationale

de la Sécurité des Systèmes d'Information", formerly known as DCSSI, in France, BSI (Federal Office for Information Security) in German, etc.

American National Standards Institute (ANSI). The ANSI X9 committee develops standards for the financial services industry. Many Elliptic curves based schemes are standardized by the X9 committee, ECDSA in ANSI X9.62 [ANS05], STS [DIF92] (see also section 4.3), ECMQV (see section 4.4), and ECIES, among others, are standardized in ANSI X9.63 [ANS01b]. The hash functions considered for the ECDSA scheme are the functions of the SHA family (SHA–1, SHA–224, SHA–256, SHA–384 and SHA–512 [FIP08]). The considered key derivation function in X9.63 is hash function based. Notice also that the domain parameters are considered only over the prime and binary fields.

National Institute of Standards and Technology (NIST). The NIST is a federal agency within the US Commerce Department's Technology Administration. Its mission includes the proposition of security related Federal Information Processing Standards (FIPS) intended for use by US federal government departments. The FIPS standards, including AES and HMAC, are probably, among the most widely adopted and deployed cryptographic schemes around the world. The ECDSA signature scheme, and many elliptic curves based cryptographic protocols, among which an UM variant termed 'dhHybrid1' in the standard (see also section 4.2), and some ECMQV variants, are standardized in [NIS07].

Institute of Electrical and Electronics Engineers (IEEE). The scope of the IEEE P1363 working group is rather large, including schemes based on integer factorization, elliptic curves, and lattices. The P1363 standard [IEE00] includes elliptic curve signature schemes (among which ECDSA), and elliptic curve key agreement schemes (ECMQV protocol, and some variants of the Elliptic Curve Diffie–Hellman (ECDH) protocol). The current draft [IEE09] includes the HMQV protocol. The P1363 differs from the ANSI and FIPS standards in that it has no mandated minimum security requirement. The considered finite field are the prime and binary fields.

Standards for Efficient Cryptography Group (SECG). This standard body, led by Certicom, tries to bring elliptic curve cryptographic techniques into real–life business. In 2000, the SECG body published two standards: the SEC 1, which standardizes some elliptic curve cryptographic schemes, and SEC 2, which recommends domain parameters to use with these schemes. While these standards were updated recently, the SECG body provides no other standard. More information about the SECG, and its two standards can be found at http://www.secg.org/.

International Organization for Standardization (ISO). The ISO organization develops standards in many fields. Indeed, there are many technical committees, dealing with broad topics. The technical committees (TC) are divided into subcommittees (SC), which in turn are divided into working groups. The ISO technical committees

and working groups subdivisions seems to follow techniques applications, rather than the techniques themselves

The cryptographic standards mainly concern the following technical committees: the technical committee 68 (Financial services) which involves 28 participating countries, and the joint technical committee (JTC) 1 (Information Technology), which is a collaboration between ISO and the International Electrotechnical Commission (IEC). Most of the work of these committees are not cryptography or security related, only few subcommittees are work on security the JTC1/SC 17, the JTC1/SC 27; and the JTC1/SC 37.

The subcommittee dealing with elliptic curve cryptography is the subcommittee SC 27 of the JTC 1. The standardized schemes include the Elliptic Curve Schnorr Digital Signature Algorithm (ISO/IEC 14888–3, 2006), Elliptic curve generation techniques (ISO/IEC 15946–5, 2009), and the ECMQV protocol (ISO/IEC 15946–3:2002).

Public–Key Cryptography Standards (PKCS). The PKCS standards are proposed by RSA Data Security to computer systems developers, the aim is to provide sufficient bases for for interoperability. Many standards are proposed and deployed (see chapter 5, for instance), however the standard dealing specifically with elliptic curves based cryptography, the PKCS #13 standard, is still under development. Its scope includes domain parameters generation and validation, key generation and validation, digital signatures, public-key encryption, key agreement, and ASN.1 syntax for parameters, keys, and schemes identification. The PKCS# 11 standard considers the use of some elliptic curves based schemes, ECDSA, ECMQV, ECDH, etc. but refers to other standards for the definition of these schemes.

2.9 Patents in Elliptic Curve Cryptography

A patent is a set of exclusive rights granted by a state or a set of states to an inventor or its assignee for a limited duration, at most 20 years in France [INP09], in exchange of a public disclosure of an invention. The patent granting procedures, and requirements placed on a patent vary from one country to another. However, in most countries, a patentee holds the right to prevent others from using or distributing in any way the patented invention without permission. The exact boundary of what is protected by a patent is given by the patent *claims*. To infringe a patent, each and every element of its claims must be present in the infringing product. Even if only a single element of a patent's claims is missing in a product, the product does not infringe the patent, except if ti is used in the product something *equivalent* to the missing elements. Two elements can be considered equivalent if (1) they perform the same function, and (2) achieve the same result, in the same way.

Even if some aspects of patent law have been harmonized internationally (through treaties or organizations such as the Patent Cooperation Treaty, or the European Patent Organization), there remains however some differences between the US and European patents.

In Europe, an invention is patentable if it is novel, and *solves a technical* problem in a non–obvious way scientific theories and mathematical methods are not then patentable (see Article 52 of the European Patent Convention at http://www.epo.org/patents/law/legal-texts/epc.html). In US, the requirements are similar, but seem less restrictive; to be patentable, an invention must be novel and not obvious (see the US Code 35, Sec. 101, at http://www.gpoaccess.gov/index.html). In Europe, when two persons apply for a patent on the same patentable invention, the first to have applied will get the patent; this holds even if the second is able to prove that he discovered the invention first. Only filing date counts in Europe. In US, if two applications interfere, it is first tried to determine the first inventor, this may include examining research logbooks, dates for prototypes, and so on; if found, he gets the patent.

Some patent related confusing around elliptic curve cryptography. Patents seems to be a main factor limiting elliptic curve cryptography implementation and deployment. For instance, elliptic curve cryptography schemes were integrated in OpenSSL, only in the version 0.9.8 in 2005 (see http://www.openssl.org/.)

Numerous ECC related patents are hold by Certicom [CER] (82 US patents, 195 US and non–US patents at january 2009 — see at http://www.certicom.com/index.php/licensing/patents-issued). It seems that Certicom is the main elliptic curve cryptography related patents holder. Surprisingly, there is no patent claims at the SECG standard website (see at http://www.secg.org/) besides that of Certicom. It is difficult to provide an exhaustive review of the patents related to elliptic curve cryptography, we only list some patents or patent applications closely related to our work (most of them can be found at http://www.freepatentsonline.com/).

Two important patents related to our work are the (most recent) US patent on MQV [LAMB07] (which includes the MQV variant using the simultaneous multiplication technique) and the European patent on HMQV [KRA08], there is also a US patent application on HMQV [KRA06], we do not know however, whether or not it was granted. A shallow review of some ECC related patents is given in Table 2.5; the symbol 🖉 indicates that the concerned document is a patent application.

As one can expect, it is difficult to determine the exact boundaries of elliptic curve cryptography related patents. Nevertheless, it seems possible to implement numerous ECC schemes without any patent infringement. An interesting indication on alternative possible implementations comes from the RSA Laboratories FAQ entries[3]: "In all of these cases, it is the implementation technique that is patented, not the prime or representation, and there are alternative, compatible implementation techniques that are not covered by the patents". Naturally, this is not legal truth; and, as often with patents, only court truth matters.

2.10 Examples of elliptic curves cryptography deployment

In this section we provide few examples of elliptic curves cryptography deployment.

[3] http://www.rsa.com/rsalabs/node.asp?id=2325

Table 2.5: Some ECC related patents

US 5761305:	"Key Agreement and Transport Protocols with Implicit Signatures" (pertain to MQV)	Certicom (Jun.98)
US 5889865:	"Key Agreement and Transport Protocol with Implicit Signatures" (pertain to MQV)	—— (Mar. 99)
US 5896455:	"Key Agreement and Transport Protocol with Implicit Signatures" (pertain to MQV)	—— (Apr. 99)
US 6122736:	"Key agreement and transport protocol with implicit signatures" (pertain to MQV)	—— (Sept. 00)
US 6785813:	"Key agreement and transport protocol with implicit signatures" (pertain to MQV)	—— (Aug. 04)
US 5933504:	"Strengthened public key protocol" (pertains to preventing the small-subgroup attacks in key agreement protocols)	—— (Aug. 99)
US 6141420:	"Elliptic Curve Encryption Systems" (pertains to point compression)	—— (Oct. 00)
US 7418099:	"Method and Apparatus for Performing Elliptic Curve Arithmetic" (pertains to preventing invalid curve attacks in key agreement protocols)	—— (Aug. 08)
US 0033405	"Method and Structure for Challenge–Response Signatures and High–Performance Secure Diffie–Hellman Protocols" (pertain to HMQV and the (X, D)CR schemes)	IBM (Aug.06)
US 5787028	" Multiple Bit Multiplier" (pertains to multiplication optimizations in $GF(2^m)$)	Certicom (Jul. 98)
US 0040225	"Fast Scalar Multiplication for Elliptic Curve Cryptosystems over Prime Fields " (pertains to point multiplication optimizations and side channel attacks resilience)	ATMEL (Feb. 2010)
US 7602907	"Elliptic Curve Point Multiplication" (pertains to point multiplication optimizations and side channel attacks resilience)	Microsoft (Oct. 09)
US 0180612	"Authentication Method Employing Elliptic Curve Cryptography" (pertains to mobile systems authentication)	Lin & Associates IP, Inc. (Jul. 2008)

With the Vista operating system, Microsoft designed a new cryptographic services provider, termed "Cryptography API: Next Generation (CNG)", for a long term replacement of the Microsoft CryptoAPI. The CNG is intended for developers in the Windows programming environment. The main add–on of the CNG compared to the Microsoft CryptoAPI is that it implements some elliptic curve based schemes, mainly the ECDSA and the ECDH. More details on the CNG can be found at http://msdn.microsoft.com/en-us/library/aa376210(v=VS.85).aspx.

Also about five years ago, Microsoft embedded a digital rights management (DRM) system into the Windows Media Player. When data representation related technicalities (which may be XML, base 64 encoding, etc., depending on the the DRM version)

are ruled out, the DRM license request protocol is simply a signed and encrypted request followed by a signed and encrypted response. The client generates an ECDSA signed license request, together with a 16–byte random key; the request is encrypted using the RC4 scheme, and the RC4 encrypting key encrypted using the sever's public key. The public key encryption scheme is ECC Elgamal. The server's response is encrypted in a similar way. More details on the windows DRM system can be found at http://msdn.microsoft.com/en-us/library/cc227964.aspx.

Elliptic curve cryptography is now used in the German e–passports (see http://www.en.bmi.bund.de). The scheme used for document signature is ECDSA; some authentication protocols are also proposed (mainly variants of the static Diffie–Hellman protocol). The PACE protocol (see section 2.6.3) is today mandatory [BSI10].

The security and efficiency advantages of elliptic curve based schemes over RSA or multiplicative discrete logarithm based schemes are clear. As security will remain an essential concern in communications, even if the deployment of ECC schemes is still limited compared to RSA, there should be no doubt that the next generation cryptographic tools will be mainly elliptic curves based.

Chapter 3

Security Models for Authenticated Key Agreement

Contents

3.1 Introduction

Much of recent research on key agreement deals with provably secure key exchange. Since this approach was pioneered by Bellare and Rogaway [BEL93a], different models were proposed [BEL95, BLA97a, SHO99, CAN01, KRA05, LAMA07]. Among these models, the Canetti–Krawczyk (CK) [CAN01] and extended Canetti–Krawczyk (eCK) [LAMA07] models (which are incomparable [CRE09b, UST09]) are considered as "advanced" approaches to capture security of key agreement protocols; and security arguments for recent protocols are usually provided in the (e)CK models.

Broadly, a security model specifies, among other things, what constitutes a security failure, and what adversarial behaviors are being protected against. The aim is that a protocol shown secure, in the model, confines to the minimum the effects of the considered adversarial behaviors. In the CK and eCK models, session specific information leakages are respectively captured using reveal queries on *session states* and *ephemeral keys*, which store session specific information; the adversary is supposed to interact with parties, and to try to distinguish a session key from a randomly chosen value. A protocol is secure if an adversary controlling communications between parties, cannot distinguish a session key from a random value, unless it makes queries which overtly reveal the session key.

While it is desirable that key agreement protocols confine the adverse effects of failures the minimum possible, some of the (e)CK–secure protocols fail to be impersonation attack resilient, when some session specific information leakage occurs [SAR09a]. In both the CK and (e)CK models, the definitions of the reveal queries make a part of practical attacks unconsidered.

In this chapter, we outline the main ideas of the original Bellare–Rogaway (BR) model upon which are based the (e)CK models, and present the CK models and eCK models. We also highlight the importance of finely understanding the limitations of the eCK models, when using them in security reductions. In section 3.6, we propose a strong security definition, from [SAR10a], which encompasses the eCK model, and provides stronger reveal queries to the adversary. To illustrate that our security definition is usable, and not too restrictive, we propose a new authenticated key agreement protocol called Strengthened MQV (SMQV), which meets our security definition under the gap Diffie–Hellman assumption and the random oracle model [SAR10a]. The SMQV protocol provides the same efficiency as the (H)MQV protocols [LAW03, KRA05b]. In addition, because of its resilience to intermediate results leakages, SMQV is particularly suited for implementations using a tamper–resistant device, to store the static keys, together with a host machine on which sessions keys are used. In such SMQV implementations, the non–idle time computational effort of the device can be securely reduced to few non–costly operations. This chapter includes results from [SAR09a, SAR09b, SAR10a].

3.2 The Bellare–Rogaway Model(s)

The first complexity theoretic formalization of a secure key exchange protocol, seems to appear in [BEL93a]. Although the original variant of the BR model covered the two party mutual authentication case in the symmetric key setting, other variants dealing with three party server–based protocols [BEL95] or public key based protocols [BLA97a], among others, was subsequently proposed. The original Bellare–Rogaway model is outlined hereunder.

The model considers a set I of parties sharing a *long–lived key generator*, i.e. a polynomial time machine, which on input the security parameter λ, outputs a long lived key for each couple $\{i, j\}$, $i, j \in I$. Each party $i \in I$, is modeled with an infinite set of oracle $\Pi^s_{i,j}$ $j \in I$, $s \in \mathbb{N}$; an oracle $\Pi^s_{i,j}$ models the s–th session that the entity i attempts to run with j. The considered adversary (denoted here $\mathcal{A}$) is a probabilistic polynomial time machine in control of communication links betweens parties. The oracles only interact with the adversary; they do not interact directly one another.

The *conversation* of an oracle is defined to be the ordered concatenation of incoming and outgoing messages. Let $\mathfrak{in}^s_{i,j}$ denote the ordered concatenation of $\Pi^s_{i,j}$'s incoming messages, and $\mathfrak{out}^s_{i,j}$ be defined in a similar way. The conversations of two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ are said to be matching if $\mathfrak{in}^s_{i,j}$ equals $\mathfrak{out}^t_{j,i}$, and conversely. The adversary is not allowed to access directly to the oracles private information; however, the following queries are allowed for oracle activations, and also to capture information leakages that may occur.

- *Send*(i, j, s, M). This query provides the oracle $\Pi^s_{i,j}$ with the message M. As an answer, the oracle returns the next message that would be returned in a normal execution of the protocol. When M is the empty string, the call is understood as an oracle (des)activation query.
- *Reveal*(i, j, s). This query captures session key leakages. When it is issued on an already completed oracle $\Pi^s_{i,j}$, the attacker is provided with the oracle's session key; if the oracle has not completed yet, the call is ignored.
- *Corrupt*(i, K). With this query, the adversary learns all long–lived secrets at the party i, and set i's long–lived secrets to K; from there, i is controlled by the adversary. (Notice that the model from [BEL93a] does not consider the *Corrupt* query; however, it is considered in security arguments using this model [BEL95, BLA97a, BLA97b, WON01, CHE03], so we describe it as part of the model.)
- *Test*(i, j, s). When a completed oracle $\Pi^s_{i,j}$ is issued with this query, it chooses $\gamma \in_R \{0, 1\}$, and provides $\mathcal{A}$ with k_γ, where k_γ is the key computed in the oracle, if γ equals 1, otherwise, a random value chosen under the distribution of session keys. The adversary then guesses the value of γ. (The adversary has to produce its guess immediately after learning k_γ; as discussed in [CAN01a, Appendix A], this requirement is not strong enough.)

A completed oracle $\Pi^s_{i,j}$ is said to be *fresh*, if it was not sent a *Reveal* query, and no completed oracle with matching conversation was sent a *Reveal* query.
A protocol is said to be (BR) *secure*, if

- in the presence of an adversary which faithfully convey messages, two oracles with matching conversations yield the same session key, and
- no polynomially bounded adversary can succeed in guessing the value of γ in a fresh oracle with probability significantly greater than $1/2$.

Blake–Wilson *et al.*'s variant. This variant introduces rather minor modifications to deal with the public key setting [BLA97a]. Except the setup phase, in which the long–lived key generator in the original model is replaced with a key pair generator, the Blake–Wilson *et al.*'s variant is mainly the same as the original one. Indeed, this variant seems to be rather an illustration of the BR model's usability in the public key setting (modulo minor modifications) than a proposition of a new model.

Shoup's Generalization. Shoup [SHO99] proposes a generalization of the BR model. The proposed model, somewhat abstract, considers the use of session keys in applications. Three corruption cases are considered. (1) In the *static corruption* case, the adversary decides about entities to corrupt, prior to starting interactions between parties. (2) In the *adaptive corruption* case, the adversary can corrupt an entity at any time. The corrupt query provides the adversary with the long–lived secrets of the entity on which the query is issued. (3) Similar to the adaptive corruption, the *strong adaptive corruption* can be issued on an entity at any time, the attacker then obtains all secret information at the party.

The Bellare–Rogaway security is shown equivalent to Shoup's simulation–based security in the static corruption case [SHO99]; this may seem surprising, as the BR model allows the attacker to corrupt parties at any moment, however as shown in [SHO99, section 15.3], in the BR model, security against static, adaptive and strong adaptive corruptions are equivalent. The security definition in the strong adaptive corruption case is shown to imply the BR security, added with forward secrecy. The strong adaptive security is not shown to imply the adaptive one.

In hindsight, while introducing a fundamental approach, upon which recent "advanced" security models are built, the Bellare–Rogaway security definition(s), seems unsatisfactory; as ignoring many important security attributes, among which key compromise impersonation resilience, ephemeral private keys leakage resilience, and in general session–specific information leakages resilience.

3.3 The Canetti–Krawczyk Model(s)

The parties considered in this model [CAN01] are probabilistic polynomial time machines, $\hat{P}_1, \cdots, \hat{P}_n$, interconnected each other. A protocol is defined as a collection of procedures run by a finite number of parties; each protocol specifies its processing rules for incoming and outgoing messages. A key exchange is a protocol which involves two parties. A *session* is an instance of a protocol run at a party. At session activation, a *session state* is created to contain specific information computed in the session.

For key exchange protocols, each session is activated with a quadruple $(\hat{P}_i, \hat{P}_j, \psi, \varsigma)$, where $\hat{P}_i$ is the session owner, $\hat{P}_j$ is the peer, ψ is the session identifier, and ς is the role of $\hat{P}_i$ in the session. The session identifier is required to be unique at each party involved in the session, i.e., a party never uses the same session identifier twice. Two sessions with initial inputs $(\hat{P}_i, \hat{P}_j, \psi, \varsigma)$ and $(\hat{P}_j, \hat{P}_i, \psi', \varsigma')$ are said to be matching if $\psi = \psi'$.

Adversary. The adversary $\mathcal{A}$ is a probabilistic polynomial time machine in control of communications between parties; outgoing messages are submitted to $\mathcal{A}$ which decides about their delivery. $\mathcal{A}$ also decides about session activations; in addition, it is given the following queries, aiming to model practical information leakages.

- *SessionStateReveal.* When this query is issued on an uncompleted session, the adversary obtains the ephemeral information contained in the session. The model does not however specify the information revealed by this query; it leaves this to be specified by each protocol.
- *SessionKeyReveal.* With this query, the adversary obtains, the session key derived in a completed and unexpired session.
- *Corrupt.* When this query is issued on a party, the adversary obtains all the information hold by the party, including its static private key and session states. Once the query is issued, the attacker (which is in control of communication links) can impersonate the party at will; one then consider the party under the

attacker's control. A party against which this query is not issued is said to be *honest*.

- *Expire.* This query models the erasure of a session key (and state) from the session owner's memory. Notice that a session can be expired while its matching session is unexpired.
- *Test.* As in the Bellare–Rogaway model, when the test query is issued on a completed (and unexpired) session, a bit γ is chosen at random, and depending on the value of γ, the attacker is provided with either the session key, or a random value chosen under the distribution of session keys. The attacker is allowed to continue its run with regular queries, but not to reveal the test session or its matching session's key or state.

Definition 6. A session is said to be *locally exposed* if it was sent a *SessionStateReveal* query, a *SessionKeyReveal* query, or if its owner is corrupted. A session is said to be *exposed* if it or its matching session is locally exposed. An unexposed session is said to be *CK–fresh*.

With this session freshness definition, a secure protocol is as follows.

Definition 7 (CK–security)**.** Let Π be a protocol such that if two honest parties complete matching sessions then, except with negligible probability, they both compute the same session key.

- Π is said to be (CK–)secure if no polynomially bounded adversary can distinguish a *CK–fresh session* key from a random value (chosen under the distribution of session keys) with probability significantly greater than 1/2.
- And Π is said to be CK–secure without *forward secrecy*[1], if it meets the CK security definition against any adversary which is not allowed to perform *Expire* queries.

It is worthwhile to mention that HMQV security arguments [KRA05] are provided in a variant of the CK model, termed CK_{HMQV}, which defines matching sessions using the matching conversations notion. Sessions are identified with quadruples $(\hat{P}_i, \hat{P}_j, X, Y)$ where $\hat{P}_i$ is the session owner, $\hat{P}_j$ is the peer, and X (resp. Y) is the outgoing (resp. incoming) ephemeral public key. This matching sessions definition is used in [KRA05] to "simplify the presentation" [KRA05, section 2, p. 10], however, this significantly changes the security definition. Also, in separate analyses, Krawczyk [KRA05, sections 6 and 7.4] allows the adversary to query a *SessionStateReveal* on the test session, or to learn the static private key of the test–session owner. Notice also that, seemingly, the purpose of [KRA05], was not to propose a new security model, as it refers to [CAN01] for details [KRA05, p. 9], and considers its session identifiers and matching sessions definition (which make the CK and CK_{HMQV} models incomparable) as consistent with the CK model [KRA05, p. 10].

[1] Some authors, [KRA05] for instance, use the term 'perfect forward secrecy', but following [BOY03], we prefer 'forward secrecy' to avoid a confusion with (Shannon's) 'perfect secrecy'.

Captured Security Attributes. CK–secure protocols provide *security against eavesdropping only attackers*, as a successful eavesdropping only attacker would succeed in distinguishing test. They provide also the *known session key security attribute*, since an adversary gains no useful information about fresh sessions by learning other session keys.

In addition, within the limits of matching sessions definition[2], CK secure protocols provide *impersonation* and *unknown key share attacks resilience*. As if an attacker was able to makes two non–matching sessions $(\hat{P}_i, \hat{P}_j, \psi, \varsigma)$ and $(\hat{P}_t, \hat{P}_j, \psi, \varsigma')$ yield the same session key, it would issue a *SessionKeyReveal* on one of the sessions and use the other as test session; *key replication attack resilience* is captured also.

The CK–security captures also forward secrecy, as if an attacker could compute a session key, using only the session owner's session key, it would succeed in distinguishing test, using the following sequence of queries: $Test(sid)$, then $Expire(sid)$, and then $Corrupt(\hat{P}_i)$ ($\hat{P}_i$ is the session owner, sid the session identifier). The attacker compute session key using $\hat{P}_i$'s static key and answers to the test query.

3.4 The Extended Canetti–Krawczyk Model

The eCK model was initially presented as a strengthening of the CK model [LAMA07]. The model focuses on the public key setting. Here also, the adversary is supposed in control of communications between parties supposed to be probabilistic polynomial time machines. All parties share a group $\mathcal{G}$, in which their static public keys are chosen. They share also a common certification authority (CA). At certificate issuance, the CA is only supposed to test the public key for membership in $\mathcal{G}^*$; no proof of possession of the corresponding private key is required.

For two–party DH protocols, sessions are activated at each party $\hat{P}_i$, with parameters $(\hat{P}_i, \hat{P}_j)$ or $(\hat{P}_i, \hat{P}_j, Y)$, which make $\hat{P}_i$ initiate a session with peer $\hat{P}_j$ or respond to a session initiated at $\hat{P}_j$. Each session at $\hat{P}_i$ is identified with a quintuple $(\hat{P}_i, \hat{P}_j, X, Y, \varsigma)$ where $\hat{P}_j$ is the peer, X is the outgoing ephemeral key, Y the incoming one, and ς the role of $\hat{P}_i$ in the session (initiator ($\mathcal{I}$) or responder ($\mathcal{R}$)). Two sessions with identifier $(\hat{P}_i, \hat{P}_j, X, Y, \varsigma)$ and $(\hat{P}_j, \hat{P}_i, Y, X, \varsigma')$ (with $\varsigma' \neq \varsigma$) are said to be matching.

A major deviation from the CK model is introduced through the *ephemeral key* notion. The ephemeral key of a session is required (1) to contain all session–specific information, i.e., all ephemeral secret information an attacker may query, and (2) all computations performed to derive the session key at a party "must deterministically depend on that party's ephemeral key, long–term secret key, and communication received from the other party" [LAMA07]. Static key leakages are captured through the *StaticKeyReveal* query, which provides the attacker with the static private key of the entity upon which it is issued. The model's queries are the following.

- *EphemeralKeyReveal(sid).* When the attacker issues this query, it is provided with the *sid* session ephemeral key.

[2] As discussed in section 3.5 some practical attacks are not considered when matching sessions are defined using matching identifiers, as in the CK model.

- *SessionKeyReveal(sid).* If the session *sid* has already completed, the session owner provides the attacker with the session key; otherwise, the query is ignored.
- *StaticKeyReveal(party).* When the attacker issues this query, it is provided with the static private key of the entity upon which it is issued.
- *EstablishParty(party).* The adversary registers a static public key on behalf of a party. As the adversary is in control of communications, from there, the party is supposed totally controlled by the adversary. A party against which this query is not issued is said to be *honest*.
- As in the CK model, a *Test(sid)* query is provided; recall that when the adversary issues this query, a bit is chosen at random, and depending on the chosen bit–value, the adversary is provided with either the *sid* session key or a random value chosen under the distribution of session keys.

With these queries, session key freshness and protocol security are defined as follows.

Definition 8 (Session freshness)**.** Let *sid* be the identifier of a session completed at an honest party $\hat{A}$ with some honest peer $\hat{B}$.

- The session *sid* is said to be *strongly eCK–fresh* if none of the following holds.
  - $\mathcal{A}$ issues a *SessionKeyReveal* query on *sid* or sid^* (if sid^* exists).
  - $\mathcal{A}$ issues a *StaticKeyReveal* query on $\hat{A}$ and an *EphemeralKeyReveal* query on *sid*.
  - sid^* exists, and $\mathcal{A}$ makes a *StaticKeyReveal* query on $\hat{B}$ and an *EphemeralKeyReveal* query on sid^*.
  - The session with identifier sid^* does not exist, and $\mathcal{A}$ makes a *StaticKeyReaveal* query on $\hat{B}$ *before* the completion of the *sid* session.
- And *sid* is said to be *eCK–fresh* if it meets the strong freshness variant, where the last condition is replaced by
  - The session with identifier sid^* does not exist and $\hat{A}$ makes a *StaticKeyReveal* query on $\hat{B}$.

Definition 9 (eCK–security)**.** Let Π be a protocol such that if two honest parties complete matching sessions, they both compute the same session key.

- The protocol Π is said to be *strongly eCK–secure* if no polynomially bounded adversary can distinguish a *strongly eCK–fresh* session key from a random value (chosen under the distribution of session keys) with probability significantly greater than 1/2.
- And Π is said to be *eCK–secure* if no polynomially bounded adversary can distinguish an *eCK–fresh* session key from a random value (chosen under the distribution of session keys) with probability significantly greater than 1/2.

The eCK model provides two security definitions, the strong security definition captures *forward secrecy*. However, as shown by Krawczyk [KRA05, section 3.2], no implicitly authenticated two–message protocol such as ours can achieve forward secrecy. These protocols, can however achieve *weak forward secrecy*, which (loosely speaking) is: session keys (previously) computed in presence of an eavesdropping only attacker cannot be recovered, when the adversary is given the session owner's static private key.

Captured Security Attributes. *If* ephemeral keys are *actually* defined to contain all information on which leakage may occur in practice[3], the eCK model can be considered as preferable to the CK one, as it captures the main security attributes captured in the CK model. In addition, the eCK model allows an attacker to issue an ephemeral key reveal query on a test session. Hence, if the ephemeral key is defined to contain the ephemeral DH exponent, the eCK model captures the (desirable) security attribute that an attacker should not be able to compute a session key, unless it knows both the static and ephemeral private keys of an entity implicated in the session.

3.4.1 The Menezes–Ustaoglu Variant

Until there, we have implicitly made the assumption that, at session activation, a party knows the identity of its peer; this is the *pre–specified peer model* [CAN02]. In the *post–specified peer model*, a party may not know the identity of its peer at session activation; the peer's identity is learned during the protocol run.

Menezes and Ustaoglu [MEN09] propose a variant of the eCK model, called *combined eCK model* (ceCK), geared to the post model ('pre–specified peer' and 'post–specified peer' are respectively shortened to 'pre' and 'post'). In this model, sessions are activated at a party $\hat{A}$ with parameters $(\hat{A}, \tilde{B})$ or $(\tilde{A}, \hat{B}, \mathtt{in})$, where $\tilde{P}$ is a *destination address* for message delivery, and $\mathtt{in}$ is the incoming message; $\hat{A}$ is the session initiator if the activation parameter is $(\hat{A}, \tilde{B})$. As in the eCK model, the ceCK matching sessions are defined using matching conversations; session identifiers are updated to contain the peer's identity once known. In addition to the eCK reveal queries, the ceCK adversary is also provided with an *EphemeralPublicKeyReveal* query. When the adversary issues this query at a party, it obtains the ephemeral public key that the party will use the next time it is activated for session initialization. Notice that the ceCK *EphemeralPublicKeyReveal* query definition seems conflicting with the common use of ephemeral public key, as an ephemeral public key is usually computed *after* a session activation.

Except these differences on session identifiers, session activation parameters, and the addition of the *EphemeralPublicKeyReveal* query, the ceCK and eCK security definitions are the same.

Equally important, the separation between the pre and post models security seems unclear. The protocol $\mathcal{P}$ claimed secure in the pre model, and not executable in the post model (unless changed in a fundamental way) [MEN09], is in effect insecure in the pre–model, if the considered security model is strong enough (see section 3.5.1). The HMQV protocol is executable in the post model, but claimed insecure (in the post–model). In fact, the proposed attack [MEN09, section 3.2] cannot be carried out in practice, not because it requires an important on–line computational effort (2^{60} operations, when the order of $\mathcal{G}$ is a 160–bit prime), but since the step (2.c) of the attack cannot be performed without changing the $\hat{M}$ found at the step (2.b). In practice, $\hat{M}$ (is a certificate, and) is defined to contain M (which is provided to the certification authority at certificate issuance), and when M is changed, so is $\hat{M}$ (notice

[3] Notice that this may differ from the information an adversary is formally allowed to query.

that changing M requires another certificate issuance); and then, after the step (2.c), the claimed equality between $\bar{H}(X, \hat{M})$ and $\bar{H}(X, \hat{B})$ does not hold.

For the Σ_0 protocol (secure in the post model, while insecure in the pre one), the model in which it is shown secure in the post model [CAN02] is not strong enough; it is not difficult to see, for instance, that the Σ_0 protocol is both eCK and ceCK insecure.

Table 3.1 gives some protocols, the security definition they meet, and the assumptions under which the security reductions are carried. All analysis are performed in the random oracle (RO) model [BEL93b]. The count of computational effort (CE) at each party is naive, i.e., without optimizations from [MEN96, Algorithm 14.88] and [MRA96], incoming key validation is not considered also.

Table 3.1: Exemples of Protocols meeting different security definitions.

Protocol	Security	Assumptions	CE
CMQV [UST08]	eCK	GDH	3
HMQV [KRA05]	CK_{HMQV}	CDH, KEA1	2.5
MQV [LAW03]	—	—	2.5
NAXOS [LAMA07]	eCK	GDH	4
NAXOS–C [MEN09]	ceCK	GDH	4
UP [UST09]	ceCK	GDH	3.5

The MQV protocols are probably the most efficient of all know two-party Diffie–Hellman protocols. However, MQV has not security reduction. The key idea in the MQV design (a dual identification scheme) is reused in the (C, H)MQV protocols, yielding efficient protocols. The computational effort for a party in the other protocols is significantly far from the 2.5 exponentiations in the (H)MQV protocols, which represents only 25% additional computational effort per party, when compared to the unauthenticated Diffie–Hellman protocol.

3.5 Security Nuances in the (e)CK Models

In this section, we discuss shades of security in the (e)CK models, which can make practical attacks unconsidered in security reductions.

3.5.1 Inadequacy of the CK Matching Sessions Definition

Recall that, in the CK model, two sessions with activation parameters $(\hat{P}_i, \hat{P}_j, s, role)$ and $(\hat{P}_j, \hat{P}_s, s', role')$ are said to be matching if they have the same identifiers $(s = s')$. The requirement about the identifiers (id) used at a party is that *"the session id's of no two KE sessions in which the party participates are identical"* [CAN01]. Session identifiers may, for instance, be nonces generated by session initiators and provided to the peers through the first message in the protocol. In this case, when each party stores the previously used identifiers and verifies at session activation that the session identifier was not used before, the requirement that a party never uses the same identifier twice is achieved.

Unfortunately, when a party, say $\hat{B}$, has no means to be aware of the sessions initiated at the other parties, and intended to it, apart from receiving the initiator's message, the CK model insufficiently captures impersonations attacks. Consider, for instance, Protocol 3.1 (wherein H and H_2 are digest functions); it is from [MEN09], and is CK–secure under the Gap Diffie–Hellman assumption [MAU96] and the Random Oracle (RO) model [BEL93b]. As the session state is defined to be the ephemeral DH exponent[4], while the protocol $\mathcal{P}$ is (formally) CK–secure, its practical security is unsatisfactory, unless session identifiers are added with further restrictions. If session identifiers are nonces generated by initiators, the protocol $\mathcal{P}$ practically fails in authentication. As an illustration, consider Attack 3.1, wherein the attacker impersonates $\hat{A}$, exploiting a knowledge of an ephemeral DH exponent used at $\hat{A}$.

Protocol 3.1 The protocol $\mathcal{P}$

PROTOCOL MESSAGES:

$\underline{\hat{A}} : a, A, \hat{A}$		$\underline{\hat{B}} : b, B, \hat{B}$
$x \in_R [1, q-1]$,		
$X = G^x$,		
$t_A = H_2(B^a, \mathcal{I}, s, \hat{A}, \hat{B}, X)$,		
	$\boxed{X,\ t_A} \longrightarrow$	
		$y \in_R [1, q-1]$,
		$Y = G^y$,
		$t_B = H_2(B^a, \mathcal{R}, s, \hat{B}, \hat{A}, Y)$,
	$\longleftarrow \boxed{Y,\ t_B}$	
$K = H(Y^x, X, Y)$		$K = H(X^y, X, Y)$,

I) At session activation with parameters $(\hat{A}, \hat{B}, s)$, $\hat{A}$ does the following:
   (a) Create a session with identifier $(\hat{A}, \hat{B}, s, \mathcal{I})$.
   (b) Choose $x \in_R [1, q-1]$.
   (c) Compute $X = G^x$ and $t_A = H_2(B^a, \mathcal{I}, s, \hat{A}, \hat{B}, X)$.
   (d) Send $(\hat{B}, \hat{A}, s, X, t_A)$ to $\hat{B}$.

II) At receipt of $(\hat{B}, \hat{A}, s, X, t_A)$, $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Create a session with identifier $(\hat{B}, \hat{A}, s, \mathcal{R})$.
   (c) Compute $\sigma = A^b$ and verify that $t_A = H_2(\sigma, \mathcal{I}, s, \hat{A}, \hat{B}, X)$.
   (d) Choose $y \in_R [1, q-1]$.
   (e) Compute $Y = G^y$, $t_B = H_2(\sigma, \mathcal{R}, s, \hat{B}, \hat{A}, Y)$, and $K = H(X^y, X, Y)$.
   (f) Send $(\hat{A}, \hat{B}, s, \mathcal{I}, Y, t_B)$ to $\hat{A}$.
   (g) Destroy y, σ, and complete $(\hat{B}, \hat{A}, s, \mathcal{R})$ by accepting K as the session key.

III) At receipt of $(\hat{A}, \hat{B}, s, \mathcal{I}, Y, t_B)$, $\hat{A}$ does the following:
   (a) Verify the existence of an active session with identifier $(\hat{A}, \hat{B}, s, \mathcal{I})$.
   (b) Verify that $Y \in \mathcal{G}^*$.
   (c) Verify that $t_B = H_2(B^a, \mathcal{R}, s, \hat{B}, \hat{A}, Y)$.
   (d) Compute $K = H(Y^x, X, Y)$.
   (e) Destroy x, and complete $(\hat{A}, \hat{B}, s, \mathcal{I})$, by accepting K as the session key.

[4] [MEN09] does not specify the information contained in a session state. But, since the adversary controls communications between parties, we do not see another non–superfluous definition of a session state which Protocol $\mathcal{P}$ can be shown CK–secure with; as the protocol is insecure if the session state is defined to be $\sigma = A^b$.

Attack 3.1 Impersonation Attack against $\mathcal{P}$ using Ephemeral DH exponent Leakage

I) At the activation of a session $(\hat{A}, \hat{B}, s, \mathcal{I})$, the attacker $\mathcal{A}$ does the following:
   (a) Intercept $\hat{A}$'s message to $\hat{B}$ $(\hat{B}, \hat{A}, s, X, t_A)$.
   (b) Perform a session *SesssionStateReveal* query on $(\hat{A}, \hat{B}, s, \mathcal{I})$ (to obtain x).
   (c) Send $(\hat{A}, \hat{B}, s, \mathcal{I}, \bar{1}, 0^{|q|})$ to $\hat{A}$, where $\bar{1}$ is the identity element in $\mathcal{G}$ and $0^{|q|}$ is the string consisting of $|q|$ zero bits (as $\bar{1} \notin \mathcal{G}^*$, $\hat{A}$ aborts the session $(\hat{A}, \hat{B}, s, \mathcal{I})$).

II) When $\mathcal{A}$ decides later to impersonate $\hat{A}$ to $\hat{B}$, it does the following:
   (a) Send $(\hat{B}, \hat{A}, s, X, t_A)$ to $\hat{B}$.
   (b) Intercept $\hat{B}$'s message to $\hat{A}$ $(\hat{A}, \hat{B}, s, \mathcal{I}, Y, t_B)$.
   (c) Compute $K = H(Y^x, X, Y)$.
   (d) Use K to communicate with $\hat{B}$ on behalf of $\hat{A}$.

The attacker makes $\hat{B}$ run a session and derive a key with the belief that its peer is $\hat{A}$; in addition, the attacker is able to compute the session key that $\hat{B}$ derives; in practice, this makes the protocol fail in authentication.

The capture of impersonation attacks based on ephemeral DH exponent leakages is insufficient in the CK–model, unless the matching sessions definition is added with further restrictions. The reason is that (in a formal analysis) the attacker $\mathcal{A}$ cannot use the session at $\hat{B}$ (in which it impersonates $\hat{A}$) as a test session, since the matching session is exposed, while there is no guarantee that (in practice) $\hat{B}$ would not run and complete such a session. If matching sessions are defined using matching conversations, it becomes clear that Protocol $\mathcal{P}$ is both formally and practically insecure. Indeed, in this case, a leakage of an ephemeral DH exponent in a session allows an attacker to impersonate indefinitely the session owner to its peer in the exposed session.

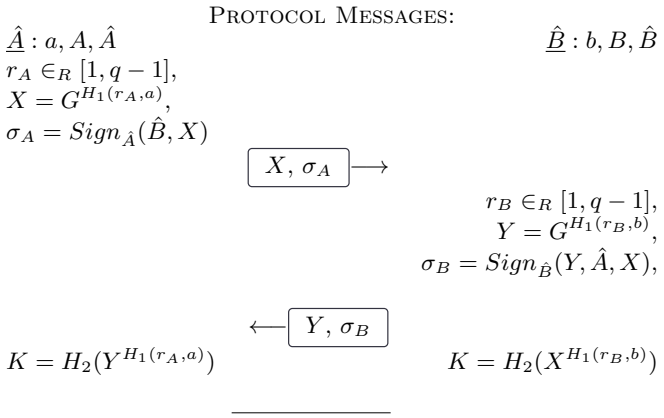
3.5.2 The eCK Ephemeral Key and the Use of the NAXOS Transformation

In the eCK model [LAMA07], the ephemeral key of a session is required to contain all session–specific information an attacker may query, and all computations performed to derive a session key have to deterministically depend on the ephemeral key, static key, and communication received from the peer.

The design and security arguments of many eCK secure protocols, among which CMQV [UST08], NAXOS(+, –C) [LAMA07, LEE08b, MEN09], and NETS [LEE08a], use the NAXOS transformation [LAMA07], which consists in defining the ephemeral DH exponent as the digest of a randomly chosen value and the static private key of the session owner, and (unnaturally) destroying it after each use. The ephemeral key (i.e., the session specific information the attacker may learn) is then defined to be the random value. And, as the attacker is not allowed to issue both an *EphemeralKeyReveal* query on a test session and a *StaticKeyReveal* query on the session owner, with this transformation designing eCK–secure protocols is relatively convenient. However, from a practical perspective, it seems difficult to see how the NAXOS transformation prevents leakages on the ephemeral DH exponents (which are not contained in ephemeral keys). How does this transformation prevent an attacker using power ana-

lysis on the exponentiation to perform, since exponentiations with the DH exponent are performed in each completed session. And, in any environment, which does not guarantee that leakages on DH exponents cannot occur, the NAXOS type protocols security is at best unspecified.

Protocol 3.2 Signed Diffie–Hellman using NAXOS transformation



I) The initiator $\hat{A}$ does the following:
   (a) Choose $r_A \in_R [1, q-1]$, compute $X = G^{H_1(r_A,a)}$, and destroy $H_1(r_A, a)$.
   (b) Compute $\sigma_A = Sign_{\hat{A}}(\hat{B}, X)$.
   (c) Send $(\hat{B}, X, \sigma_A)$ to $\hat{B}$.

II) $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Verify that σ_A is a valid signature with respect to $\hat{A}$'s public key and message $(\hat{B}, X)$.
   (c) Choose $r_B \in_R [1, q-1]$, compute $Y = G^{H_1(r_B,b)}$, and destroy $H_1(r_B, b)$.
   (d) Compute $\sigma_B = Sign_{\hat{B}}(Y, \hat{A}, X)$;
   (e) Send $(Y, \hat{A}, X, \sigma_B)$ to $\hat{A}$.
   (f) Compute $K = H_2(X^{H_1(r_B,b)})$.

III) $\hat{A}$ does the following:
   (a) Verify that $Y \in \mathcal{G}^*$.
   (b) Verify that σ_B is a valid signature with respect to $\hat{B}$'s public key and message $(Y, \hat{A}, X)$.
   (c) Compute $K = H_2(Y^{H_1(r_A,a)})$.

IV) The shared session key is K.

Consider, for instance, Protocol 3.2, it is from an earlier version[5] of [CRE09b]. If the ephemeral keys are defined to be r_A and r_B (as in the NAXOS security arguments [LAMA07]) and the signature scheme is secure against chosen message attacks, Protocol 3.2 can be shown eCK–secure. Nevertheless, Protocol 3.2 is insecure if the ephemeral key is defined to contain the ephemeral DH exponent. In fact, as shown in Attack 3.2, an adversary which (partially[6]) learns $H_1(r_A, a)$ in a session between

[5] http://eprint.iacr.org/cgi-bin/versions.pl?entry=2009/253, version 20090625.

[6] If the adversary partially learns $H_1(r_A, a)$, it recovers the remaining part, using Shanks' Baby Step Giant Step algorithm [TES01a] or Pollard's rho algorithm [TES01a], if the bits it learns are the most significant ones, or tools from [GOP07] if the leakage is on middle–part bits; recovering $H_1(r_A, a)$

$\hat{A}$ and $\hat{B}$, initiated by $\hat{A}$, can indefinitely impersonate $\hat{A}$ to $\hat{B}$ [SAR10b, SAR10c]. One can see also that the NAXOS protocol [LAMA07] cannot meet the eCK–security definition, if the ephemeral key is defined to contain the ephemeral DH exponent.

Attack 3.2 Impersonation Attack against SDHNT using Ephemeral DH exponent Leakage

(1) $\mathcal{A}$ records $\hat{A}$'s outgoing message, say $(\hat{B}, X^{(l)}, \sigma_A^{(l)})$, together with the learned ephemeral DH exponent in the leaked session.
(2) Each time $\mathcal{A}$ decides to impersonate $\hat{A}$ to $\hat{B}$, it does the following:
  (a) Send $(\hat{B}, X^{(l)}, \sigma_A^{(l)})$ to $\hat{B}$.
  (b) Intercept $\hat{B}$'s message to $\hat{A}$ $(Y, \hat{A}, X^{(l)}, \sigma_B)$.
  (c) Compute $K = H_2(Y^{H_1(r_A, a)})$.
  (d) Use K to communicate with $\hat{B}$ on behalf of $\hat{A}$.

Session key derivation generally involves some intermediate results (the values a session owner may need to compute or store between messages), which cannot be computed, given only the session's ephemeral private key. For instance, in the protocols using the NAXOS transformation, ephemeral DH exponents cannot be computed given only the ephemeral key (the random nonce), in the Protocols 1 and 2 from [KIM09, pp. 6, 12] (which do not use the NAXOS transformation), an attacker cannot learn the intermediate value $s_1 = x + a_1$ or $s_2 = x + a_2$. In fact, in the eCK model, once the ephemeral key is defined, the parties are considered as black boxes, which may only leak session keys, ephemeral keys, and static keys. This does not match the usual protocol implementations, wherein a party may store, and leak intermediate secret values between messages. And, in any environment, which does not guarantee that leakages on intermediate secret values cannot occur, the concrete security of the eCK–secure protocols is unspecified.

3.6 Stronger Security

In this section, we describe the strengthened eCK model [SAR10a, SAR10c], which considers leakages on intermediate results (the values a party may need to compute between messages or before a session key), encompasses the eCK model [LAMA07], and provides stronger reveal queries to the attacker.

A common setting wherein key agreement protocols are often implemented is that of a server used together with a (computationally limited) tamper–resistant device, which stores the long–lived secrets. In such a setting, safely reducing the non–idle time computational effort of the device, is usually crucial for implementation efficiency. To reduce the device's non–idle time computational effort, ephemeral keys can be computed on the device in idle–time, or on the host machine when the implemented protocol is ephemeral DH exponent leakage resilient.

In many DH protocols, (C, FH, H)MQV–C [LAW03, UST08, SAR09a, KRA05] and NAXOS(+, –C) [LEE08b, MEN09, LAMA07], for instance, the computation of

from partial leakage requires some additional computations.

the intermediate results is more costly than that of the ephemeral public key. For these protocols, implementation efficiency is significantly enhanced when the ephemeral keys are computed on the device in idle–time, while the intermediate results, which require expensive on–line computations and session keys are computed on the host machine. Unfortunately the security of the (e)CK–secure protocols, when leakages on the intermediate results are considered is at best unspecified. A security definition which captures attacks based on intermediate result leakages is clearly desirable. The model we propose captures such attacks, together with the attacks captured in the (e)CK models.

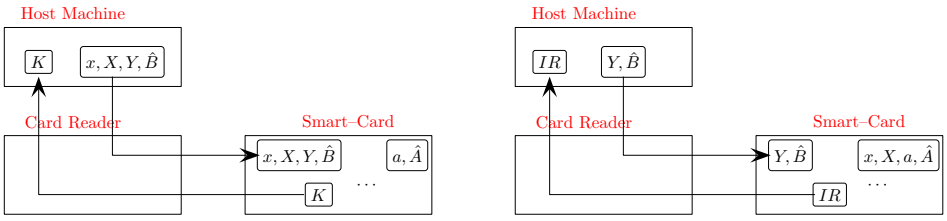
Session. We suppose $n \leqslant \mathcal{L}(|q|)$ (for some polynomial $\mathcal{L}$) parties $\hat{P}_{i=1,\cdots,n}$ supposed to be probabilistic polynomial time machines and a certification authority (CA) trusted by all parties. The CA is only required to verify that public keys are valid ones (i.e., public keys are only tested for membership in $\mathcal{G}^*$; no proof of possession of corresponding private key is required). Each party has a certificate binding its identity to its public key. A session is an instance of the considered protocol, run at a party. A session at $\hat{A}$ (with supposed peer $\hat{B}$) can be created with parameter $(\hat{A}, \hat{B})$ or $(\hat{B}, \hat{A}, m)$, where m is an incoming message, supposed to be from $\hat{B}$; $\hat{A}$ is the initiator if the creation parameter is $(\hat{A}, \hat{B})$, otherwise a responder. At session activation, a session state is created to contain the information specific to the session. Each session is identified with a tuple $(\hat{P}_i, \hat{P}_j, \mathfrak{out}, \mathfrak{in}, \varsigma)$, wherein $\hat{P}_i$ is the session holder, $\hat{P}_j$ is the intended peer, $\mathfrak{out}$ and $\mathfrak{in}$ are respectively the concatenation of the messages $\hat{P}_i$ sends to $\hat{P}_j$, or supposes to be from $\hat{P}_j$, and ς is $\hat{P}_i$'s role in the session (initiator or responder). Two sessions with identifiers $(\hat{P}_i, \hat{P}_j, \mathfrak{out}, \mathfrak{in}, \varsigma)$ and $(\hat{P}'_j, \hat{P}'_i, \mathfrak{out}', \mathfrak{in}', \varsigma')$ are said to be matching if $\hat{P}_i = \hat{P}'_i$, $\hat{P}'_j = \hat{P}'_j$, $\varsigma \neq \varsigma'$, and at completion $\mathfrak{in} = \mathfrak{out}'$ and $\mathfrak{out} = \mathfrak{in}'$.

For the two–pass DH protocols, each session is denoted with an identifier $(\hat{A}, \hat{B}, X, \star, \varsigma)$, where $\hat{A}$ is the session holder, $\hat{B}$ is the peer, X is the outgoing message, ς indicates the role of $\hat{A}$ in the session (initiator ($\mathcal{I}$) or responder ($\mathcal{R}$)), and $\star$ is the incoming message Y if it exists, otherwise a special symbol meaning that an incoming message is not received yet; in that case, when $\hat{A}$ receives the incoming public key Y, the session identifier is updated to $(\hat{A}, \hat{B}, X, Y, \varsigma)$. Two sessions with identifiers $(\hat{A}, \hat{B}, X, Y, \mathcal{I})$ and $(\hat{B}, \hat{A}, Y, X, \mathcal{R})$ are said to be matching. Notice that the session matching $(\hat{B}, \hat{A}, Y, X, \mathcal{R})$ can be any session $(\hat{A}, \hat{B}, X, \star, \mathcal{I})$; as $X, Y \in_R \mathcal{G}^*$, a session cannot have (except with negligible probability) more than one matching session.

Adversary and Security. The adversary $\mathcal{A}$, is a probabilistic polynomial time machine; outgoing messages are submitted to $\mathcal{A}$ for delivery ($\mathcal{A}$ decides about messages delivery). $\mathcal{A}$ is also supposed to control session activations at each party via the *Send($\hat{P}_i, \hat{P}_j$)* and *Send($\hat{P}_j, \hat{P}_i, Y$)* queries, which make $\hat{P}_i$ initiate a session with peer $\hat{P}_j$, or respond to the (supposed) session $(\hat{P}_j, \hat{P}_i, Y, \star, \mathcal{I})$. We suppose that the considered protocol is implemented at a party following one of the approaches hereunder. We suppose also that at each party an untrusted host machine is used together with a tamper–resistant device. Notice that basing our model on these implemen-

tation approaches does not make it specific; rather, this reduces the gap that often exists between formal and practical security. Such modeling techniques, which take into account hardware devices and communication flows between components, were previously used in [BRE02].

Approach 1. In this approach, the static keys are stored on the device (a smart–card, for instance) the ephemeral keys are computed on the host machine, passed to the smart–card together with the incoming public keys; the device computes the session key, and provides it to the host machine (application) for use. The information flow between the device and the host machine is depicted in Figure (3.1a). This implementation approach is safe for eCK–secure protocols when ephemeral keys are defined to be ephemeral DH exponents, as a leakage on an ephemeral DH exponent does not compromise the session in which it is used. In addition, when an attacker learns a session key, it gains no useful information about the other session keys.



(a) Implementation Approach 1 *(b)* Implementation Approach 2

Figure 3.1: Implementation Approaches

Approach 2. Another approach, which has received less attention in the formal treatment of DH protocols, is when the ephemeral keys, and top level intermediate results are computed on the device, and the host machine is provided with some intermediate results IR which it computes the session key with. As the computation of the intermediate results is often more costly than that of the ephemeral public keys, implementation efficiency is often significantly enhanced using this approach. Naturally, this comes with the requirement that leakages on the intermediate results should not compromise any unexposed session; and whatever intermediate results an attacker learns, it should not be able to impersonate a party, unless it knows the party's static private key. Namely, an adversary may have a malware running on the host machine at a party, and learn all values computed or used at the party, except those stored in the party's tamper–proof device; this should not compromise any unexposed session.

We define two sets of queries, modeling leakages that may occur on either implementation approaches. We consider leakages on ephemeral and static private keys, and also on any intermediate (secret) value which evaluation requires some secret information. As the adversary can compute any information which evaluation requires only public information, considering leakages on such data is superfluous.
In Set 1, which models leakages in the first implementation approach, the following

queries are allowed.

- *EphemeralKeyReveal(session)*: this query models leakages on ephemeral DH exponents.
- *Corrupt$_{SC}$(party)*: this query models an attacker which (bypasses the eventual tamper protection mechanisms on the device, and) gains read access to the device's private memory; it provides the attacker with the device owner's static private key.
- *SessionKeyReveal(session)*: when the attacker issues this query on an already completed session, it is provided with the session key.
- *EstablishParty(party)*: with this query, the adversary registers a static key on behalf of a party; as the adversary controls communications, from there the party is supposed totally controlled by $\mathcal{A}$. A party against which this query is not issued is said to be *honest*.

In Set 2, which models leakages on the second implementation approach, the following queries are allowed; the definitions remain unchanged for the queries belonging also to Set 1.

- For any node in the *intermediate results,* which computation requires a secret value, a reveal query is defined to allow leakage on the information computed in this node. These queries models leakages that may occur on intermediate results in computing session keys.
- *SessionKeyReveal(session)*.
- *EstablishParty(party)*.
- *Corrupt$_{SC}$(party)*.

Before defining the seCK security, we define the session freshness notion. Test queries can only be performed on fresh sessions.

Definition 10 (Session Freshness [SAR10a, SAR10c])**.** Let Π be a protocol, and $\hat{A}$ and $\hat{B}$ two honest parties, *sid* the identifier of a completed session at $\hat{A}$ with peer $\hat{B}$, and *sid*$'$ the matching session's identifier. The session *sid* is said to be *locally exposed* if one of the following holds.

- $\mathcal{A}$ issues a *SessionKeyReveal* query on *sid*.
- The implementation at $\hat{A}$ follows the first approach and $\mathcal{A}$ issues an *EphemeralKeyReveal* query on *sid* and a *Corrupt$_{SC}$* query on $\hat{A}$.
- The implementation at $\hat{A}$ follows the second approach and $\mathcal{A}$ issues an intermediate result query on *sid*.

The session *sid* is said to be *exposed* if (1) it is locally exposed, or (2) its matching session *sid*$'$ exists and is locally exposed, or (3) the session with identifier *sid*$'$ does not exist and $\mathcal{A}$ issues a *Corrupt$_{SC}$* query on $\hat{B}$. An *unexposed* session is said to be *fresh*.

Our session freshness conditions match exactly the intuition of the sessions one may hope to protect. In particular, it lowers (more than in the eCK model) the necessary adversary restrictions for any reasonable security definition. Notice that only the queries corresponding to the implementation approach followed by a party can be issued on it.

Definition 11 (Strengthened eCK–Security [SAR10a, SAR10c])**.** Let Π be a protocol, such that if two honest parties complete matching sessions, then they both compute the same session key. The protocol Π is said to be *seCK–secure*, if no polynomially bounded adversary can distinguish a fresh session key from a random value, chosen under the distribution of session keys, with probability significantly greater than 1/2.

Forward Secrecy. As shown in [KRA05], no implicitly authenticated two–message protocol such as ours can achieve *forward secrecy*. Indeed, our security definition captures *weak forward secrecy*, which (loosely speaking) is: any session established without an active involvement of the attacker remains secure, even when the implicated parties static keys are disclosed. The seCK security definition can be completed with the session key expiration notion [CAN01] to capture forward secrecy. Although the protocol we propose can be added with a third message, and yield a protocol which (provably) provides forward secrecy, in the continuation, we work with the security definition without forward secrecy, and focus on two–pass DH protocols.

3.7 Relations between the seCK and eCK models

In the eCK model, an adversary may compromise the ephemeral key, static key, or session key at a party, independently of the way the protocol is implemented. The seCK model considers an adversary which may (have a malware running at a party's host machine and) learn all information at the party, except those stored in a tamper–resistant device. The seCK approach seems more prevalent in practice, and reduces the gap that often exists between formal arguments and practical implementations security.

The eCK and seCK session identifiers and matching sessions definitions are the same. When the adversary issues the $Corrupt_{SC}$ query at a party, it is provided with the party's static key; the $Corrupt_{SC}$ query is the same as the eCK *StaticKeyReveal* query. For a session between two parties, say $\hat{A}$ and $\hat{B}$, following the first implementation approach, the seCK session *freshness* definition reduces to the eCK freshness with ephemeral keys defined to be the ephemeral DH exponents. By assuming that all parties follow the first implementation approach, the seCK–security definition reduces to the eCK one; the seCK model encompasses the eCK one.

Proposition 2. *Any seCK–secure protocol is also an eCK–secure one.*

The seCK model also separates clearly from the eCK model. The eCK model does not consider leakages on intermediate results; and this makes many of the eCK secure protocols insecure in the seCK model. For instance, in the CMQV protocol (shown eCK–secure) [UST08], an attacker which learns an ephemeral secret exponent in a session, can indefinitely impersonate the session owner; the same holds for the (H)MQV(–C) protocols (see sections 4.5 and 4.6). It is not difficult to see that NAXOS cannot meet the seCK security definition. The protocols 1 and 2 from [KIM09, pp. 6, 12] (shown eCK–secure) fail in authentication when leakages one the intermediate results are considered. Indeed an attacker, which learns the ephemeral secret exponents

$s_1 = x + a_1$ and $s_2 = x + a_2$ in a session at $\hat{A}$, can *indefinitely impersonate* $\hat{A}$ *to any party*. Notice that the attacker cannot compute $\hat{A}$'s static key from s_1 and s_2, while it is not difficult to see that leakages on s_1 (or s_2) *and* the ephemeral key, in the *same session* imply $\hat{A}$'s static key disclosure.

Also, the seCK model is not only about the formal treatment of key agreement protocols, it is also about securely implementing authenticated key exchange. When a protocol is shown secure in the seCK model, it is also clearly specified how it can be *securely implemented.* In fact, providing security arguments in the seCK model, includes (1) stating which operations or data have to be handled in an area with the same protection mechanisms as the area in which is stored the session owner's static key, and (2) stating which information can be computed or stored in an untrusted area. Notice that the existence of protection mechanisms for a static private key is inherent to the ability to keep secret a static secrete information; this is a prerequisite for cryptography.

The seCK model is practically stronger than the CK model [CAN01]. Key Compromise Impersonation resilience, for instance, is captured in the seCK model while not in CK model. As shown in [CHO05], and illustrated in section 4.6 with Protocol $\mathcal{P}$, the CK model is enhanced when matching sessions are defined using matching conversations. In addition, the seCK reveal query definitions go beyond the usual CK session state definition (ephemeral DH exponents). Compared to the CK_{HMQV} model[7] [KRA05], the reveal query definitions are enhanced in the seCK model to capture attacks based on intermediate result leakages. In the HMQV security arguments [KRA05, subsection 7.4], the session state is defined to contain the ephemeral DH exponent[8]; the HMQV protocol does not meet the seCK–security.

3.8 The Strengthened MQV Protocol

In this section, we present the *strengthened* MQV protocol, and its building blocks, to show that the seCK security definition is useful, and not limiting; as seCK–secure protocols can be built with usual building blocks. We start with the following variants of the FXCR and FDCR signature schemes (see section 4.7).

Definition 12 (FXCR–1 Signature)**.** Let $\hat{B}$ be a party with public key $B \in \mathcal{G}^*$, and $\hat{A}$ a verifier; $\hat{B}$'s signature on a message m and challenge X provided by $\hat{A}$ ($x \in_R [1, q-1]$ is chosen and kept secret by $\hat{A}$) is $Sig_{\hat{B}}(m, X) = (Y, X^{s_B})$, where $Y = G^y$, $y \in_R [1, q-1]$ is chosen by $\hat{B}$, and $s_B = ye + b$, where $e = \bar{H}(Y, X, m)$. And, $\hat{A}$ accepts the pair (Y, σ_B) as a valid signature if $Y \in \mathcal{G}^*$ and $(Y^e B)^x = \sigma_B$.

The security of the FXCR–1 schemes can be shown with arguments similar to that of the FXCR scheme, given in section 4.7.1.

[7] CK_{HMQV} is the variant of the CK model in which the HMQV security arguments are provided; however, it seems that the aim of [KRA05] was not to propose a new model, as it refers to [CAN01] for details [KRA05, p. 9], and considers its session identifiers and matching sessions definition (which make the CK and CK_{HMQV} models incomparable) as consistent with the CK model [KRA05, p. 10]. See [CRE09b] for a comparison between the CK_{HMQV} and (e)CK models.

[8] In [KRA05, subsection 5.1], the session state is defined to contain the ephemeral public keys, but this definition is superfluous, as the adversary controls communications between parties.

Proposition 3 (FXCR–1 Security). *Under the CDH assumption in $\mathcal{G}$ and the RO model, there is no adaptive probabilistic polynomial time attacker, which given a public key B, a challenge X_0 ($B, X_0 \in_R \mathcal{G}^*$), together with hashing and signing oracles, outputs with non–negligible success probability a triple (m_0, Y_0, σ_0) such that:*
(1) (Y_0, σ_0) is a valid signature with respect to the public key B, and the message–challenge pair (m_0, X_0); and
(2) (Y_0, σ_0) was not obtained from the signing oracle with a query on (m_0, X_0).

The dual variant of the FXCR–1 scheme is as follows.

Definition 13 (FDCR–1 Scheme). Let $\hat{A}$ and $\hat{B}$ be two parties with public keys $A, B \in \mathcal{G}^*$, and m_1, m_2 two messages. The dual signature of $\hat{A}$ and $\hat{B}$ on the messages m_1, m_2 is $DSig_{\hat{A},\hat{B}}(m_1, m_2, X, Y) = (X^d A)^{ye+b} = (Y^e B)^{xd+a}$, where $X = G^x$ and $Y = G^y$ are chosen respectively by $\hat{A}$ and $\hat{B}$, $d = \bar{H}(X, Y, m_1, m_2)$, and $e = \bar{H}(Y, X, m_1, m_2)$.

As for the FXCR–1 scheme, the following proposition can be shown with arguments similar to that of the FDCR scheme 4.7.2.

Proposition 4 (FDCR–1 Security). *Let $A = G^a, B, X_0 \in_R \mathcal{G}^*$ ($A \neq B$). Under the RO model, and the CDH assumption in $\mathcal{G}$, given a, A, B, X_0, a message m_{1_0}, a hashing oracle, together with a signing oracle (simulating $\hat{B}$'s role), no adaptive probabilistic polynomial time attacker can output, with non–negligible success probability a triple (m_{2_0}, Y_0, σ_0) such that:*
(1) $DSig_{\hat{A},\hat{B}}(m_{1_0}, m_{2_0}, X_0, Y_0) = \sigma_0$; and
(2) (Y_0, σ_0) was not obtained from the signing oracle with a query on some (m'_1, X') such that $X_0 = X'$ and $(m'_1, m'_2) = (m_{1_0}, m_{2_0})$, where m'_2 is a message returned at signature query on (m'_1, X'); (m_{1_0}, m_{2_0}) denotes the concatenation of m_{1_0} and m_{2_0}.

The strengthened MQV protocol follows from the FDCR–1 scheme; a run of SMQV is as in Protocol 3.3. The execution aborts if any verification fails. The shared secret σ is the FDCR–1 signature of $\hat{A}$ and $\hat{B}$, on challenges X, Y and messages $\hat{A}, \hat{B}$ (the representation of $\hat{A}$ and $\hat{B}$'s identities). The parties identities and ephemeral keys are used in the final digest computation to make the key replication resilience security attribute immediate (and also to avoid unknown key share attacks). A run of SMQV requires 2.5 times a single exponentiation (2.17 times a single exponentiation when the multiple exponentiation technique [MEN96, Algorithm 14.88] is used); this efficiency equals that of the remarkable (H, FH)MQV protocols. SMQV provides all the security attributes of the (C, H)MQV protocols, added with ephemeral secret exponent leakage resilience.

Moreover, suppose an implementation of SMQV or (C, H)MQV using an untrusted[9] host machine together with a computationally limited tamper resistant device; and suppose that the session keys are used by some applications running on the host machine, and that the ephemeral keys are computed on the device in idle–time. This idle–time pre–computation seems common in practice [SCH91] (and possible in both

[9]There are many reasons for not trusting the host machine: bogus or trojan software, viruses, etc.

Protocol 3.3 The Strengthened MQV Protocol

PROTOCOL MESSAGES:

$\underline{\hat{A}} : a, A, \hat{A}$	$\underline{\hat{B}} : b, B, \hat{B}$
$x \in_R [1, q-1]$,	
$X = G^x$,	
$(\hat{A}, \hat{B}, X) \longrightarrow$	
	$y \in_R [1, q-1]$,
	$y = G^y$,
$\longleftarrow (\hat{B}, \hat{A}, Y)$	
$s_A = (xd + a) \bmod q$,	$s_B = (ye + b) \bmod q$,
$\sigma_A = (Y^e B)^{s_A}$	$\sigma_B = (X^d A)^{s_B}$,
$K = H(\sigma_A, \hat{A}, \hat{B}, X, Y)$	$K = H(\sigma_B, \hat{A}, \hat{B}, X, Y)$.

I) The initiator $\hat{A}$ does the following:
   (a) Choose $x \in_R [1, q-1]$.
   (b) Compute $X = G^x$.
   (c) Send $(\hat{A}, \hat{B}, X)$ to the peer $\hat{B}$.

II) At receipt of $(\hat{A}, \hat{B}, X)$, $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Choose $y \in_R [1, q-1]$.
   (c) Compute $Y = G^y$.
   (d) Send $(\hat{B}, \hat{A}, Y)$ to $\hat{A}$.
   (e) Compute $d = \bar{H}(X, Y, \hat{A}, \hat{B})$ and $e = \bar{H}(Y, X, \hat{A}, \hat{B})$.
   (f) Compute $s_B = ye + b \bmod q$ and $\sigma = (X^d A)^{s_B}$.
   (g) Compute $K = H(\sigma, \hat{A}, \hat{B}, X, Y)$.

III) At receipt of $(\hat{B}, \hat{A}, Y)$, $\hat{A}$ does the following:
   (a) Verify that $Y \in \mathcal{G}^*$.
   (b) Compute $d = \bar{H}(X, Y, \hat{A}, \hat{B})$ and $e = \bar{H}(Y, X, \hat{A}, \hat{B})$.
   (c) Compute $s_A = xd + a \bmod q$, and $\sigma = (Y^e B)^{s_A}$.
   (d) Compute $K = H(\sigma, \hat{A}, \hat{B}, X, Y)$.

IV) The shared session key is K.

the (C, H)MQV and SMQV protocols). But, as (C, H)MQV(–C) is not ephemeral secret exponent leakage resilient (see section 4.6), the ephemeral secret exponents ($s_A = x + da$ or $s_B = y + eb$) cannot be used on the untrusted host machine. The exponentiation $\sigma = (YB^e)^{s_A} = (XA^d)^{s_B}$ has to be performed on the device *in non idle–time*. In contrast, for SMQV, $\sigma = (Y^e B)^{s_A} = (X^d A)^{s_B}$ can be computed on the host machine, after the ephemeral secret exponent is computed on the device. Because the session key is used on the host machine, and a leakage of only the ephemeral secret exponent, in a SMQV session, does not compromise any other session; there is no need to protect the ephemeral secret exponent more than the session key. In SMQV, the non–idle time computational effort of the device reduces to few non–costly operations (one integer addition, one integer multiplication, and one digest computation), while for (C, H)MQV at least one exponentiation has to be performed on the device in non idle–time.

Table 3.2 summarizes the comparisons between SMQV and some other DH pro-

Table 3.2: Security and Efficiency Comparison between SMQV and other DH protocols.

Protocol	Security	Assumptions	NC	NICE 1	NICE 2
CMQV [UST08]	eCK	GDH	3E	1E	1E
FHMQV [SAR09a]	CK_{FHMQV}	GDH	2.5E	1E	1D + 1A + 1M
HMQV [KRA05]	CK_{HMQV}	GDH, KEA1	2.5E	1E	1E
MQV [LAW03]	–	–	2.5E	1E	1E
NAXOS [LAMA07]	eCK	GDH	4E	3E	3E
NAXOS–C [MEN09]	ceCK	GDH	4E	3E	3E
SMQV [SAR10c]	seCK	GDH	2.5E	1E	1D + 1A + 1M

tocols. All the security reductions are performed using the Random Oracle model [BEL93b]; incoming ephemeral keys are validated[10]. KEA1 stands for "Knowledge of Exponent Assumption" [BEL04], CDH and GDH stand respectively for "Computational DH" and "Gap DH" assumptions [OKA01]. The *'A'*, *'D'*, *'E'*, and *'M'* stand respectively for *integer* a*ddition,* d*igest computation,* e*xponentiation, and integer* m*ultiplication.* The NC column indicates the *n*aive *c*ount efficiency (i.e., without optimizations from [MEN96, Algorithm 14.88] or [MRA96]); NICE 1 and NICE 2 indicate the *n*on–*i*dle time *c*omputational *e*ffort of the device in the two approaches (when ephemeral keys are computed in idle–time).

The MQV protocol has no security reduction[11]. The FHMQV security arguments were initially provided in a model which considers intermediate results and ephemeral key leakages in two separate settings; the model implicitly assumes that all parties follow the same implementation approach, and cannot be shown to encompass the CK or eCK models. (However, FHMQV can be shown to meet the seCK security definition — the protocol is discussed in section 4.7.) In contrast, the seCK model considers also the security of sessions between parties following different implementation approaches, and its matching sessions definition makes it encompass the eCK model. The CMQV and NAXOS protocols are shown eCK–secure, they both use the NAXOS transformation. The NAXOS–C security arguments are provided in the *combined eCK model* (ceCK) [MEN09], geared to the post–specified peer model.

3.9 Security Analysis of the SMQV Protocol

The SMQV protocol provides more security attributes than the NAXOS(+, –C), (C, H)MVQ protocols, in addition to allow particularly efficient implementations, in environments wherein a tamper proof device is used to store private keys.

[10] Ephemeral key validation is voluntarily omitted in the HMQV design [KRA05], but the HMQV protocol is known to be insecure if ephemeral keys are not validated [MEN06].

[11] We are aware of [KUN06], which shows that (under the RO model and the CDH assumption) the MQV variant wherein d and e are computed as $\bar{H}(X)$ and $\bar{H}(Y)$, is secure in a model of their own design. Notice that, for this variant, an attacker which finds $x_0 \in [1, q-1]$ such that $\bar{H}(G^{x_0}) = 0$, can impersonate *any* party to *any other* party. Finding such an x_0 requires $O(2^l)$ digest computations (see section 4.5).

Theorem 3. *Let* $s_A = xd + a$ *and* $\sigma = (Y^e B)^{s_A}$, *where* $d = \bar{H}(X, Y, \hat{A}, \hat{B})$ *and* $e = \bar{H}(Y, X, \hat{A}, \hat{B})$, *be the intermediate results in a session at* $\hat{A}$ *with peer* $\hat{B}$. *Under the GDH assumption in* $\mathcal{G}$ *and the RO model, the SMQV protocol is seCK–secure.*

In accordance with our security model, the following session activation queries are allowed.

- *Send*$(\hat{A}, \hat{B})$, which makes $\hat{A}$ perform the step I) of Protocol 3.3, and create a session with identifier $(\hat{A}, \hat{B}, X, \star, \mathcal{I})$.
- *Send*$(\hat{A}, \hat{B}, X)$, which makes $\hat{B}$ perform the step II) of Protocol 3.3, and create a session with identifier $(\hat{B}, \hat{A}, Y, X, \mathcal{R})$.
- *Send*$(\hat{A}, \hat{B}, X, Y)$, which makes $\hat{A}$ update the session identifier $(\hat{A}, \hat{B}, X, \star, \mathcal{I})$ (if any) to $(\hat{A}, \hat{B}, X, Y, \mathcal{I})$ and perform the step III) of SMQV.

The queries in Set 1 are the following: *EphemeralKeyReveal,* $Corrupt_{SC}$, *SessionKeyReveal,* and *EstablishParty.* In Set 2, the allowed queries are:

(a) $Corrupt_{SC}$, to obtain the static private key of a party;
(b) *SessionKeyReveal*, to obtain a session key;
(c) *SecretExponentReveal*, to obtain a secret exponent $s = xd + a$ or $ye + b$;
(d) *SessionSignatureReveal*, to obtain a session signature σ;
(e) *EstablishParty(party)* to register a static public key on behalf of a party.

Recall that an algorithm is said to be a *Decisional Diffie–Hellman Oracle* (DDHO) if on input $G, X = G^x, Y = G^y$, and Z chosen uniformly at random in $\mathcal{G}$, it outputs 1 if and only if $Z = G^{xy}$. And the Gap DH (GDH) assumption [OKA01] is said to hold in $\mathcal{G}^*$ if given a DDHO, there is no polynomially bounded algorithm, which solves the CDH problem in $\mathcal{G}$, with non–negligible success probability.

3.9.1 Proof of Theorem 3.

It is immediate from the definition of SMQV that if two honest parties complete matching sessions, they compute the same session key. Suppose that $\mathcal{A}$ succeeds with probability significantly greater than 1/2 in distinguishing a fresh session key from a random value. Distinguishing a fresh session key from a random value can be performed only in one of the following ways.

Guessing attack: $\mathcal{A}$ guesses correctly the test session key.

Key replication attack: $\mathcal{A}$ succeeds in making two non–matching sessions yield the same session key, it then issues a session key reveal query on one of the sessions, and uses the other as test session.

Forging attack: $\mathcal{A}$ computes the session signature σ, and issues a digest query to get the session key.

Under the RO model, guessing and key replications attacks cannot succeed, except with negligible probability. (Key replication attacks cannot succeed, as if $X \neq X'$, or $Y \neq Y'$, or $\hat{A} \neq \hat{A}'$, or $\hat{B} \neq \hat{B}'$, the probability that $H(\sigma, \hat{A}, \hat{B}, X, Y)$ equals $H(\sigma', \hat{A}', \hat{B}', X', Y')$ is negligible.) We thus suppose that $\mathcal{A}$ succeeds with non–negligible probability in forging attack. Let E be the event "$\mathcal{A}$ succeeds in forging the signature of some fresh session (that we designate by $sid_0 = (\hat{A}, \hat{B}, X_0, Y_0, \varsigma)$)." The event E

divides in E.1: "$\mathcal{A}$ succeeds in forging the signature of a fresh with matching session," and E.2: "$\mathcal{A}$ succeeds in forging the signature of a fresh without matching session." It suffices to show that neither E.1 nor E.2 can occur with non–negligible probability. Recall that a function $\varepsilon(\cdot)$ is said to be negligible if for all $c > 0$, there is some k_c such that for all $k > k_c$, $||\varepsilon(k)|| < k^{-c}$.

Analysis of E.1

Suppose that E.1 occurs with non–negligible probability; at least one of the following events occurs with non–negligible probability.
E.1.1: "E.1 $\wedge$ both $\hat{A}$ and $\hat{B}$ follow the first implementation approach";
E.1.2: "E.1 $\wedge$ both $\hat{A}$ and $\hat{B}$ follow the second implementation approach";
E.1.3: "E.1 $\wedge$ $\hat{A}$ and $\hat{B}$ follow different implementation approaches."
We have to show that none of E.1.1, E.1.2 and E.1.3 can occur, except with negligible probability.

Analysis of E.1.1. Since the test session is required to be fresh, the strongest queries that $\mathcal{A}$ can perform on $\hat{A}$, $\hat{B}$, the test session, and its matching session are (*i*) $Corrupt_{SC}$ queries on both $\hat{A}$ and $\hat{B}$; (*ii*) *EphemeralKeyReveal* queries on both sid_0 and sid'_0; (*iii*) a $Corrupt_{SC}$ query on $\hat{A}$ and an *EphemeralKeyReveal* query on sid'_0; (*iv*) an *EphemeralKeyReveal* query on sid_0 and a $Corrupt_{SC}$ query on $\hat{B}$. It thus suffices to show that none of the following events can occur with non–negligible probability.

E.1.1.1: "E.1.1 $\wedge$ $\mathcal{A}$ issues $Corrupt_{SC}$ queries on both $\hat{A}$ and $\hat{B}$";
E.1.1.2: "E.1.1 $\wedge$ $\mathcal{A}$ issues *EphemeralKeyReveal* queries on both sid_0 and sid'_0";
E.1.1.3: "E.1.1 $\wedge$ $\mathcal{A}$ issues a $Corrupt_{SC}$ query on $\hat{A}$ and an *EphemeralKeyReveal* query on sid'_0";
E.1.1.4: "E.1.1 $\wedge$ $\mathcal{A}$ issues an *EphemeralKeyReveal* query on sid_0 and a $Corrupt_{SC}$ query on $\hat{B}$."

Since from any polynomial time machine, which succeeds in E.1.1 and performs weaker queries, one can build a polynomial time machine which succeeds with the same probability, and performs one the strongest allowed queries.

Event E.1.1.1. Suppose that E.1.1.1 occurs with non–negligible probability, using $\mathcal{A}$ we build a polynomial time CDH solver $\mathcal{S}$, which succeeds with non–negligible probability. The solver interacts with $\mathcal{A}$ as follows.

(1) $\mathcal{S}$ simulates $\mathcal{A}$'s environment, with n parties $\hat{P}_1, \cdots, \hat{P}_n$, and assigns to each $\hat{P}_k$ a random static key pair $(p_k, P_k = G^{p_k})$, together with an implementation approach indication. We only suppose that the number of parties following the first implementation approach is $n_1 \geqslant 2$. $\mathcal{S}$ starts with two empty digest records $\mathfrak{H}_1$ and $\mathfrak{H}_2$. Since $\mathcal{A}$ is polynomial (in $|q|$), we suppose that each party is activated at most m times ($m, n \leqslant \mathcal{L}(|q|)$ for some polynomial $\mathcal{L}$). $\mathcal{S}$ chooses $i, j \in_R \{k \mid \hat{P}_k$ follows the first implementation approach$\}$, and $t \in_R [1, m]$ (with these choices, $\mathcal{S}$ is guessing the test session). We refer to $\hat{P}_i$ as $\hat{A}$ and $\hat{P}_j$ as $\hat{B}$.

(2) At $\bar{H}$ digest query on some $\varrho = (X, Y, \hat{P}_l, \hat{P}_m)$, $\mathcal{S}$ answers as follows: if there exists some d such that (ϱ, d) already belongs to $\mathfrak{H}_1$, $\mathcal{S}$ returns d; else, $\mathcal{S}$ provides $\mathcal{A}$ with $d \in_R \{0,1\}^l$, and appends (ϱ, d) to $\mathfrak{H}_1$.

(3) At H digest query on some $\psi = (\sigma, \hat{P}_l, \hat{P}_m, X, Y)$, $\mathcal{S}$ responds as follows: if (ψ, κ) already belongs to $\mathfrak{H}_2$, for some κ, $\mathcal{S}$ returns κ; else, $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, provides $\mathcal{A}$ with κ, and appends (ψ, κ) to $\mathfrak{H}_2$.

(4) At $Send(\hat{P}_l, \hat{P}_m)$ query, $\mathcal{S}$ chooses $x \in_R [1, q-1]$, creates a session with identifier $(\hat{P}_l, \hat{P}_m, X, \star, \mathcal{I})$, and provides $\mathcal{A}$ with the message $(\hat{P}_l, \hat{P}_m, X)$.

(5) At $Send(\hat{P}_m, \hat{P}_l, Y)$ query, $\mathcal{S}$ chooses $x \in_R [1, q-1]$, creates a session with identifier $(\hat{P}_l, \hat{P}_m, X, Y, \mathcal{R})$, provides $\mathcal{A}$ with the message $(\hat{P}_l, \hat{P}_m, X)$, and completes the session $(\hat{P}_l, \hat{P}_m, X, Y, \mathcal{R})$ ($\mathcal{S}$ also updates $\mathfrak{H}_1$ and $\mathfrak{H}_2$ in this step).

(6) At $Send(\hat{P}_l, \hat{P}_m, X, Y)$ query, $\mathcal{S}$ updates the identifier $(\hat{P}_l, \hat{P}_m, X, \star, \mathcal{I})$ (if any) to $sid = (\hat{P}_l, \hat{P}_m, X, Y, \mathcal{I})$. If the sid' session exists and is already completed, $\mathcal{S}$ sets the sid session key to that of sid'. Else, if a digest query was previously issued on some $\psi = (\sigma, \hat{P}_l, \hat{P}_m, X, Y)$, and if σ is the sid session signature ($\mathcal{S}$ can compute the session signature), $\mathcal{S}$ sets the session key to $H(\psi)$. Else, $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, sets the session key to κ, and updates $\mathfrak{H}_2$.

(7) If $\mathcal{A}$ issues a $Corrupt_{SC}$, an *EphemeralKeyReveal*, a *SessionKeyReveal*, or an *EstablishParty* query at a party following the first implementation approach, $\mathcal{S}$ answers faithfully.

(8) If $\mathcal{A}$ issues a $Corrupt_{SC}$, a *SessionKeyReveal*, a *SecretExponentReveal*, a *SessionSignatureReveal*, or an *EstablishParty* query at a party following the second implementation approach, $\mathcal{S}$ answers faithfully.

(9) At the activation of the t–th session at $\hat{A}$, if the peer is not $\hat{B}$, $\mathcal{S}$ aborts; else, $\mathcal{S}$ provides $\mathcal{A}$ with $(\hat{A}, \hat{B}, X_0)$ (recall that $\mathcal{S}$ takes as input X_0 and $Y_0 \in_R \mathcal{G}^*$).

(10) At the activation of the session matching the t–th session at $\hat{A}$, $\mathcal{S}$ provides $\mathcal{A}$ with $(\hat{B}, \hat{A}, Y_0)$.

(11) In any of the following situations, $\mathcal{S}$ aborts.
- $\mathcal{A}$ halts with a test session different from the t–th session at $\hat{A}$.
- $\mathcal{A}$ issues a *SessionKeyReveal* or an *EphemeralKeyReveal* query on the t–th session at $\hat{A}$ or its matching session.
- $\mathcal{A}$ issues an *EstablishParty* query on $\hat{A}$ or $\hat{B}$.

(12) If $\mathcal{A}$ provides a guess σ_0 of the test session signature, $\mathcal{S}$ outputs

$$\begin{aligned}\left(\sigma_0 (X_0^{d_0} A)^{-b} Y_0^{-ae_0}\right)^{(d_0 e_0)^{-1}} &= \left((X_0^{d_0} A)^{y_0 e_0} Y_0^{-ae_0}\right)^{(d_0 e_0)^{-1}} \\ &= \left((Y_0^{e_0})^{x_0 d_0 + a} Y_0^{-ae_0}\right)^{(d_0 e_0)^{-1}}\end{aligned}$$

as a guess for $CDH(X_0, Y_0)$. Otherwise $\mathcal{S}$ aborts.

The simulated environment is perfect except with negligible probability; and if $\mathcal{A}$ is polynomial, so is $\mathcal{S}$. When $\mathcal{A}$ activates the test session and its matching session, the ephemeral keys X_0 and Y_0 it is provided with are chosen uniformly at random in $\mathcal{G}^*$; their distribution is the same as that of the real X and Y. The probabil-

ity of guessing correctly the test session is $(n_1^2 m)^{-1}$; and if $\mathcal{S}$ guesses correctly the test session and E.1.1.1 occurs, $\mathcal{S}$ does not abort. Thus $\mathcal{S}$ succeeds with probability $(n_1^2 m)^{-1} \Pr(\mathsf{E.1.1.1})$ which is non–negligible, unless $\Pr(\mathsf{E.1.1.1})$ is negligible. This shows that under the CDH assumption and RO model, E.1.1.1 cannot occur, except with negligible probability.

Event E.1.1.2. If E.1.1.2 occurs with non–negligible probability, using $\mathcal{A}$, we build a polynomial time CDH solver, which succeeds with non–negligible probability. For this purpose, we modify the simulation in the analysis of E.1.1.1 as follows.

- $\mathcal{S}$ takes as input $A, B \in_R \mathcal{G}^*$.
- $\hat{A}$ and $\hat{B}$'s public keys are set to A and B; the corresponding private keys are unknown. ($\mathcal{S}$ also keeps a list of the completed session identifiers together with the session keys.)
- At $Send(\hat{P}_m, \hat{P}_l, Y)$ query, with $\hat{P}_l = \hat{A}$ or $\hat{B}$, $\mathcal{S}$ responds as follows.
  - $\mathcal{S}$ chooses $x \in_R [1, q-1]$, computes $X = G^x$, creates a session with identifier $sid' = (\hat{P}_l, \hat{P}_m, X, Y, \mathcal{R})$, and provides $\mathcal{A}$ with the message $(\hat{P}_l, \hat{P}_m, X)$.
  - $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, $d, e \in_R \{0,1\}^l$ and sets $\bar{H}(X, Y, \hat{P}_m, \hat{P}_l) = d$, $\bar{H}(Y, X, \hat{P}_m, \hat{P}_l) = e$, and the sid' session key to κ.
- At $Send(\hat{P}_l, \hat{P}_m, X, Y)$ query, with $\hat{P}_l = \hat{A}$ or $\hat{B}$, $\mathcal{S}$ does the following.
  - $\mathcal{S}$ updates the session identifier $(\hat{P}_l, \hat{P}_m, X, \star, \mathcal{I})$ (if any) to $sid = (\hat{P}_l, \hat{P}_m, X, Y, \mathcal{I})$.
  - And, (*i*) if a value is already assigned to the sid' session key, $\mathcal{S}$ sets the sid session key to that of sid'. (*ii*) Else, if a digest query was previously issued on some $\psi = (\sigma, \hat{P}_l, \hat{P}_m, X, Y)$, and if $\sigma = CDH(X^d P_l, Y^e P_m)$ (in this case, d and e are already defined, and the verification is performed using the DDHO), $\mathcal{S}$ sets the sid session key to $H(\psi)$. (*iii*) Else, $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, and sets the sid session key to κ; if no value was previously assigned to $h_1 = \bar{H}(X, Y, \hat{P}_l, \hat{P}_m)$ (resp. $h_2 = \bar{H}(Y, X, \hat{P}_l, \hat{P}_m)$), $\mathcal{S}$ chooses $d \in_R \{0,1\}^l$ and sets $h_1 = d$ (resp. $h_2 = d$).
- At $\mathcal{A}$'s digest query on $\psi = (\sigma, \hat{P}_l, \hat{P}_m, X, Y)$, with $\hat{P}_l = \hat{A}$ or $\hat{B}$, or $\hat{P}_m = \hat{A}$ or $\hat{B}$, $\mathcal{S}$ responds as follows.
  - If there is some κ such that (ψ, κ) already belongs to $\mathfrak{H}_2$, $\mathcal{S}$ returns κ.
  - Else, (*i*) if there is an already completed session with identifier $sid = (\hat{P}_l, \hat{P}_m, X, Y, \mathcal{I})$ or sid', and if $\sigma = CDH(X^d P_l, Y^e P_m)$, then $\mathcal{S}$ returns the completed session's key. (*ii*) Else, $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, sets $H(\psi) = \kappa$, and provides $\mathcal{A}$ with κ; if no value was previously assigned to $h_1 = \bar{H}(X, Y, \hat{P}_l, \hat{P}_m)$ (resp. $h_2 = \bar{H}(Y, X, \hat{P}_l, \hat{P}_m)$), $\mathcal{S}$ chooses $d \in_R \{0,1\}^l$ and sets $h_1 = d$ (resp. $h_2 = d$).
- When $\mathcal{A}$ activates the t–th session at $\hat{A}$, if the peer is not $\hat{B}$, $\mathcal{S}$ aborts; else $\mathcal{S}$ chooses $x_0 \in_R [1, q-1]$, and provides $\mathcal{A}$ with the message $(\hat{A}, \hat{B}, X_0 = G^{x_0})$.
- When $\mathcal{A}$ activates the session matching the t–th session at $\hat{A}$, $\mathcal{S}$ chooses $y_0, \in_R [1, q-1]$, and provides $\mathcal{A}$ with $(\hat{B}, \hat{A}, Y_0 = G^{y_0})$.
- If $\mathcal{A}$ issues an *EphemeralKeyReveal* query on the t–th session at $\hat{A}$ or its matching session, $\mathcal{S}$ answers faithfully.

- $\mathcal{S}$ aborts in any of the following situations:
  - $\mathcal{A}$ halts with a test session different from the t–th session at $\hat{A}$;
  - $\mathcal{A}$ issues a *SessionKeyReveal* query on the t–th session at $\hat{A}$ or its matching session;
  - $\mathcal{A}$ issues a $Corrupt_{SC}$ or an *EstablishParty* query on $\hat{A}$ or $\hat{B}$;
- If $\mathcal{A}$ halts with a guess σ_0 fo the test session signature, $\mathcal{S}$ outputs a guess of $CDH(A, B)$ from σ_0, x_0, y_0, d_0, and e_0.

Under the RO model, the simulation remains perfect, except with negligible probability. And, if E.1.1.2 occurs with non–negligible probability, $\mathcal{A}$ succeeds with non–negligible probability under this simulation. If $\mathcal{A}$ succeeds and $\mathcal{S}$ guesses correctly the test session (this happens with probability $(n_1^2 m)^{-1} \Pr(\mathsf{E.1.1.2})$), $\mathcal{S}$ outputs $CDH(A, B)$. Under the GDH assumption and the RO model, E.1.1.2 cannot occur, unless with negligible probability.

Events E.1.1.3 and E.1.1.4. The roles of $\hat{A}$ and $\hat{B}$ in E.1.1.3 and E.1.1.4 are symmetrical; it then suffices to discuss E.1.1.3. If E.1.1.3 occurs with non–negligible probability, using $\mathcal{A}$, we build a polynomial time CDH solver which succeeds with non–negligible probability. We modify the simulation in the analysis if E.1.1.1 as follows.

- $\mathcal{S}$ takes as input $X_0, B \in_R \mathcal{G}^*$.
- $\hat{B}$'s public key is set to B (the corresponding private key is unknown), and $\hat{A}$'s key pair is $(a = p_i, G^a), p_i \in_R [1, q-1]$.
- At $Send(\hat{P}_m, \hat{B}, X)$ query, $\mathcal{S}$ responds as follows. (*i*) $\mathcal{S}$ chooses $y \in_R [1, q-1]$, computes $Y = G^y$, creates a session with identifier $sid' = (\hat{B}, \hat{P}_m, Y, X, \mathcal{R})$, and provides $\mathcal{A}$ with the message $(\hat{B}, \hat{P}_m, Y)$. (*ii*) $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, $d, e \in_R \{0,1\}^l$, sets the sid' session key to κ, $\bar{H}(X, Y, \hat{P}_m, \hat{B}) = d$, and $\bar{H}(Y, X, \hat{P}_m, \hat{B}) = e$.
- At $Send(\hat{B}, \hat{P}_m, Y, X)$ query:
  - $\mathcal{S}$ updates the session identifier $(\hat{B}, \hat{P}_m, Y, \star, \mathcal{I})$ (if any) to $sid = (\hat{B}, \hat{P}_m, Y, X, \mathcal{I})$.
  - And, (*i*) if a value is already assigned to the sid' session key, $\mathcal{S}$ sets the sid session key to that of sid'. (*ii*) Else, if a digest query was previously issued on some $\psi = (\sigma, \hat{B}, \hat{P}_m, Y, X)$ (in this case, d and e are defined) and if $\sigma = CDH(X^d P_m, Y^e B)$, $\mathcal{S}$ sets the sid session key to $H(\psi)$. (*iii*) Else, $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$ and sets the sid session key to κ; if no value was previously assigned to $h_1 = \bar{H}(Y, X, \hat{B}, \hat{P}_m)$ (resp. $h_2 = \bar{H}(X, Y, \hat{B}, \hat{P}_m)$), $\mathcal{S}$ chooses $d \in_R \{0,1\}^l$ and sets $h_1 = d$ (resp. $h_2 = d$).
- At $\mathcal{A}$'s digest query on some $\psi = (\sigma, \hat{P}_l, \hat{P}_m, X, Y)$, with $\hat{P}_l = \hat{B}$ or $\hat{P}_m = \hat{B}$, $\mathcal{S}$ responds as follows. (*i*) If the same query was previously issued, $\mathcal{S}$ returns the previously returned value. (*ii*) Else, if there is an already completed session with identifier $sid = (\hat{P}_l, \hat{P}_m, X, Y, \mathcal{I})$ or sid', and if $\sigma = CDH(X^d P_l, Y^e P_m)$, $\mathcal{S}$ returns the completed session's key. (*iii*) Else, $\mathcal{S}$ chooses $\kappa \in_R \{0,1\}^\lambda$, sets $H(\psi) = \kappa$, and provides $\mathcal{A}$ with κ. If no value was previously assigned to $h_1 = \bar{H}(X, Y, \hat{P}_l, \hat{P}_m)$ (resp. $h_2 = \bar{H}(Y, X, \hat{P}_l, \hat{P}_m)$), $\mathcal{S}$ chooses $d \in_R \{0,1\}^\lambda$ and

sets $h_1 = d$ (resp. $h_2 = d$).

- When $\mathcal{A}$ activates the t–th session at $\hat{A}$, if the peer is not $\hat{B}$, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ provides $\mathcal{A}$ with $(\hat{A}, \hat{B}, X_0)$ (recall the solver takes as input X_0 and B).
- When $\mathcal{A}$ activates the session matching the t–th session at $\hat{A}$, $\mathcal{S}$ chooses $y_0 \in_R [1, q-1]$, and provides $\mathcal{A}$ with $(\hat{B}, \hat{A}, Y_0)$.
- If $\mathcal{A}$ issues an *EphemeralKeyReveal* query on the session matching the t–th session at $\hat{A}$, $\mathcal{S}$ answers faithfully.
- In any of the following situations, $\mathcal{S}$ aborts.
  - $\mathcal{A}$ halts with a test session different from the t–th session at $\hat{A}$.
  - $\mathcal{A}$ issues a $Corrupt_{SC}$ query on $\hat{B}$ or an *EstablishParty* query on $\hat{A}$ or $\hat{B}$.
  - $\mathcal{A}$ issues an *EphemeralKeyReveal* query on the t–th session at $\hat{A}$.
- If $\mathcal{A}$ halts with a guess σ_0, $\mathcal{S}$ produces $\left(\sigma_0 (X_0^{d_0} A)^{-y_0 e_0} B^{-a}\right)^{e_0^{-1}}$ as a guess for $CDH(X_0, B)$.

The simulation remains perfect, except with negligible probability; the solver $\mathcal{S}$ guesses correctly the test session with probability $(n_1^2 m)^{-1}$. If $\mathcal{A}$ succeeds under this simulation, and $\mathcal{S}$ guesses correctly the test session, $\mathcal{S}$ outputs $CDH(X_0, B)$. Hence if $\mathcal{A}$ succeeds with non–negligible probability in E.1.1.3, $\mathcal{S}$ outputs with non–negligible probability $CDH(X_0, B)$, contradicting the GDH assumption.

None of the events E.1.1.1, E.1.1.2, E.1.1.3 or E.1.1.4 can occur with non–negligible probability; E.1.1 cannot occur, unless with negligible probability.

Analysis of E.1.2. Suppose that E.1.2 occurs with non–negligible probability, we derive from $\mathcal{A}$ a polynomial time CDH solver which succeeds with non–negligible probability. The strongest queries that $\mathcal{S}$ can issue on $\hat{A}$, $\hat{B}$, the test session and its matching session are $Corrupt_{SC}$ queries on both $\hat{A}$ and $\hat{B}$. (Recall that both $\hat{A}$ and $\hat{B}$ follow the second approach). We modify the simulation in the analysis of E.1.1.1 as follows.

- $\mathcal{S}$ takes $X_0, Y_0 \in_R \mathcal{G}^*$ as input.
- $\mathcal{A}$'s environment, is simulated in the same way as in the analysis of E.1.1.1, except that i and j are chosen in $\{k \mid \hat{P}_k$ follows the second implementation approach$\}$ (we suppose here that $n - n_1 \geqslant 2$, and still refer to $\hat{P}_i$ as $\hat{A}$ and $\hat{P}_j$ as $\hat{B}$).
- $\mathcal{S}$ aborts in the following situations.
  - $\mathcal{A}$ issues an *EstablishParty* query on $\hat{A}$ or $\hat{B}$.
  - $\mathcal{A}$ halts with a test session different from the t–th session at $\hat{A}$.
  - $\mathcal{A}$ issues a *SessionKeyReveal*, a *SecretExponentReveal*, or a *SessionSignatureReveal* query on the test session or its matching session.

The simulation remains prefect, and if $\mathcal{A}$ is polynomial, so is $\mathcal{S}$. In addition, $\mathcal{S}$ guesses correctly the test session with probability $((n - n_1)^2 m)^{-1}$; and if $\mathcal{A}$ succeeds and $\mathcal{S}$ guesses correctly the test session, it outputs $CDH(X_0, Y_0)$ (from $\mathcal{A}$'s forgery a, b, d_0 and e_0). $\mathcal{S}$ succeeds with probability $((n - n_1)^2 m)^{-1} \Pr(\mathsf{E.1.2})$ which is non–

negligible, unless Pr(E.1.2) is negligible. Under the GDH assumption and the RO model, E.1.2 cannot occur with non–negligible probability.

Analysis of E.1.3. In E.1.3 ($\hat{A}$ and $\hat{B}$ follow different implementation approaches), either $\hat{A}$ or $\hat{B}$ follows the first implementation approach; we suppose that $\hat{A}$ follows the first implementation approach. (As the test session's matching session exists, from any polynomial time machine which succeeds in E.1.3 when $\hat{A}$ follows the first approach, one can derive a polynomial time machine which succeeds with the same probability when $\hat{A}$ follows the second approach.) The strongest queries that $\mathcal{A}$ can perform on $\hat{A}$, $\hat{B}$, the test session, and its matching session are (*i*) $Corrupt_{SC}$ queries on both $\hat{A}$ and $\hat{B}$, (*ii*) an *EphemeralKeyReveal* query on the test session and a $Corrupt_{SC}$ query on $\hat{B}$. And, since from any polynomial time machine which succeeds in E.1.3, and issues weaker queries, one can build a polynomial time machine which succeeds with the same probability and performs one of the above strongest queries, it suffices to consider the following events.
E.1.3.1: "E.1.3 $\wedge$ $\mathcal{A}$ issues $Corrupt_{SC}$ queries on both $\hat{A}$ and $\hat{B}$";
E.1.3.2: "E.1.3 $\wedge$ $\mathcal{A}$ issues an *EphemeralKeyReveal* query on the test session and a $Corrupt_{SC}$ query on $\hat{B}$."
To show that E.1.3.1 cannot occur with non–negligible probability, we use the simulation in the analysis of E.1.1.1, modified as follows.

- The environment remains the same except that $i \in_R \{k \mid \hat{P}_k$ follows the first implementation approach$\}$, and $j \in_R \{k \mid \hat{P}_k$ follows the second implementation approach$\}$.
- $\mathcal{S}$ aborts in any of the following situations.
  - $\mathcal{A}$ halts with a test session different from the t–th session at $\hat{A}$.
  - $\mathcal{A}$ issues a *SessionKeyReveal* query on the t–th session at $\hat{A}$ or its matching session.
  - $\mathcal{A}$ issues a *SecretExponentReveal*, or a *SessionSignatureReveal* query on the session matching the test session, or an *EphemeralKeyReveal* query on the test session.
  - $\mathcal{A}$ issues an *EstablishParty* query on $\hat{A}$ or $\hat{B}$.

Using the same arguments, as in the analysis of E.1.1.1, $\mathcal{S}$ is a polynomial time CDH solver which succeeds with probability $(n_1(n-n_1)m)^{-1}\Pr(\text{E.1.3.1})$. Under the GDH assumption and the RO model, Pr(E.1.3.1) is negligible.

Making $\mathcal{S}$ take as input $X_0, B \in_R \mathcal{G}^*$ (the arguments are similar to that used in the analysis of the event E.1.1.3), one can show also that E.1.3.2 cannot occur, unless with negligible probability.

Analysis of E.2

Suppose that E.2 ($\mathcal{A}$ succeeds in forging the signature of some fresh session without matching session) occurs with non negligible probability. As E.2 divides in
E.2.1: "E.2 $\wedge$ both $\hat{A}$ and $\hat{B}$ follow the first implementation approach";
E.2.2: "E.2 $\wedge$ both $\hat{A}$ and $\hat{B}$ follow the second implementation approach";

E.2.3: "E.2 $\wedge$ $\hat{A}$ and $\hat{B}$ follow different implementation approaches";
at least one of the events E.2.1, E.2.2, or E.2.3 occurs with non–negligible probability.

Event E.2.1. The strongest queries that $\mathcal{A}$ can perform in E.2.1 are either an *EphemeralKeyReveal* query on the test session, or a $Corrupt_{SC}$ query on $\hat{A}$. It then suffices to discuss E.2.1.1: "E.2.1 $\wedge$ $\mathcal{A}$ performs a $Corrupt_{SC}$ query on $\hat{A}$," and E.2.1.2: "E.2.1 $\wedge$ $\mathcal{A}$ performs an *EphemeralKeyReveal* query on the test session."

E.2.1.1. To show that E.2.1.1 cannot happen with non–negligible probability, we modify the simulation in the analysis of E.1.1.3 to take as input $a \in_R [1, q-1]$ and $X_0, B \in_R \mathcal{G}^*$ ($\hat{A}$'s key pair is set to (a, G^a), and $\hat{B}$'s public key to B); $\mathcal{S}$ aborts if $\mathcal{A}$ activates a session matching the t–th session at $\hat{A}$. The simulation remains perfect, except with negligible probability. And if $\mathcal{S}$ guesses correctly the test session, and $\mathcal{A}$ succeeds with a forgery σ_0, $\mathcal{S}$ outputs σ_0 as a FDCR–1 forgery, on messages $\hat{A}$ and $\hat{B}$ with respect to the public keys A and B. $\mathcal{S}$ succeeds with probability $((n-n_1)^2 m)^{-1} \Pr(\mathsf{E.2.1.1})$, and contradicts Proposition 4, unless $\Pr(\mathsf{E.2.1.1})$ is negligible.

E.2.1.2. We modify here the simulation in the analysis of E.1.1.2 to abort if $\mathcal{A}$ activates a session matching the t–th session at $\hat{A}$. The simulated environment remains perfect, except with negligible probability. And from any valid forgery σ_0, and a correct guess of the test session, $\mathcal{S}$ outputs $A^{y_0 e_0 + b}$ (from σ_0, x_0, d_0, and e_0). $\mathcal{S}$ is polynomial; and if E.2.1.2 occurs with non–negligible probability, on input $A, B \in_R \mathcal{G}^*$, $\mathcal{S}$ outputs Y_0 and $A^{y_0 e_0 + b}$ with non–negligible probability. Hence, using the "oracle replay technique" [POI00], $\mathcal{S}$ yields a polynomial time CHD solver, which succeeds with non–negligible probability; contradicting the GDH assumption.

Event E.2.2. Suppose that E.2.2 occurs with non–negligible probability, using $\mathcal{A}$, we build a polynomial time FXCR–1 signature forger, which succeeds with non–negligible probability. For this purpose, we modify the simulation in the analysis of E.1.1.1 as follows. (Notice that $\mathcal{A}$'s $Corrupt_{SC}$ queries on $\hat{A}$ can be answered faithfully.)

- $\mathcal{S}$ takes as input $X_0, B \in_R \mathcal{G}^*$.
- Both $i, j \in_R \{k \mid \hat{P}_k$ follows the second implementation approach$\}$; $\hat{A}$'s key pair is set to $(a = p_i, G^{p_i})$, $p_i \in_R [1, q-1]$ and $\hat{B}$'s public key to B; the corresponding private key is unknown (we suppose that $\hat{A} \neq \hat{B}$).
- At $Send(\hat{P}_l, \hat{B}, X)$ query, $\mathcal{S}$ answers as follows.
  - $\mathcal{S}$ chooses $s_B \in_R [1, q-1], d \in_R \{0,1\}^l$, and sets $Y = (G^{s_B} B^{-1})^{d^{-1}}$. If there is some d' such that $((X, Y, \hat{P}_l, \hat{B}), d')$ already belongs to $\mathfrak{H}_1$, $\mathcal{S}$ aborts; else, $\mathcal{S}$ appends $((X, Y, \hat{P}_l, \hat{B}), d)$ to $\mathfrak{H}_1$.
  - $\mathcal{S}$ creates a session with identifier $sid' = (\hat{B}, \hat{P}_l, Y, X, \mathcal{R})$, completes the sid' session, and provides $\mathcal{A}$ with the message $(\hat{B}, \hat{P}_l, Y)$. (Notice that $\mathcal{S}$ can compute the session signature.)
- At $\mathcal{A}$'s $Send(\hat{B}, \hat{P}_l)$ query, $\mathcal{S}$ responds as follows.
  - $\mathcal{S}$ chooses $s_B \in_R [1, q-1], e \in_R \{0,1\}^l$, and sets[12] $Y = (G^{s_B} B^{-1})^{e^{-1}}$. If

[12] To simulate consistently the intermediate values leakage in sessions at $\hat{B}$, $\mathcal{S}$ has to assign values to $\bar{H}$ query with a partially unknown input. For these queries, random values are chosen in $\{0,1\}^l$ as $\bar{H}(Y, \star, \hat{B}, \hat{P}_l)$; when $\mathcal{S}$ is queried later with $\bar{H}(Y, X, \hat{B}, \hat{P}_l)$, it responds with $\bar{H}(Y, \star, \hat{B}, \hat{P}_l)$.

there exists some X and e' such that $((Y, X, \hat{B}, \hat{P}_l,), e')$ already belongs to $\mathfrak{H}_1$, $\mathcal{S}$ aborts.
  - $\mathcal{S}$ creates a session with identifier $(\hat{B}, \hat{P}_l, Y, \star, \mathcal{I})$, and provides $\mathcal{A}$ with $(\hat{B}, \hat{P}_l, Y)$.
- When $\mathcal{A}$ activates the t–th session at $\hat{A}$, if the peer is not $\hat{B}$, $\mathcal{S}$ aborts; else, $\mathcal{S}$ provides $\mathcal{A}$ with $(\hat{A}, \hat{B}, X_0)$.
- $\mathcal{S}$ aborts in any of the following situations.
  - $\mathcal{A}$ activates at $\hat{B}$ a session matching the t–th session at $\hat{A}$.
  - $\mathcal{A}$ halts with a test session different from the t–th session at $\hat{A}$.
  - $\mathcal{A}$ issues a $Corrupt_{SC}$ query on $\hat{B}$, or an *EstablishParty* query on $\hat{A}$ or $\hat{B}$.
  - $\mathcal{A}$ issues a *SecretExponentReveal*, a *SessionSignatureReveal*, or a *SessionKeyReveal* query on the t–th session at $\hat{A}$.
- If $\mathcal{A}$ halts with a guess σ_0 of the test session signature, $\mathcal{S}$ outputs $(\sigma_0(Y_0^{e_0}B)^{-a})^{d_0^{-1}} = X_0^{y_0e_0+b}$ as a guess for a FXCR–1 forgery on challenge X_0 and message $(\hat{A}, \hat{B})$ (the concatenation of $\hat{A}$ and $\hat{B}$) with respect to the public key B.

Under the RO model, the simulation of $\mathcal{A}$'s environment is perfect, except with negligible probability. The deviation happens when the same Y is chosen twice as outgoing ephemeral key in sessions at $\hat{B}$, with the same peer $\hat{P}_l$, this happens with probability less than m/q (which is negligible). Hence, under this simulation E.2.2 occurs with non–negligible probability. And, when $\mathcal{A}$ outputs a correct forgery, and $\mathcal{S}$ guesses correctly the test session, $\mathcal{S}$ outputs a valid FXCR–1 signature forgery on challenge X_0 and message $(\hat{A}, \hat{B})$ with respect to the public key B. $\mathcal{S}$ succeeds with probability $((n - n_1)^2 m)^{-1} \Pr(\mathsf{E.2.2})$, where negligible terms are ignored, contradicting Proposition 3.

Event E.2.3. The test session's matching session does not exist, and $\hat{A}$ and $\hat{B}$ follow different implementation approaches.

- If $\hat{A}$ follows the first implementation approach (E.2.3.1), $\mathcal{A}$ is allowed to issue either a $Corrupt_{SC}$ query on $\hat{A}$, or an *EphemeralKeyReveal* query on the test session.
  - If E.2.3.1.1: "E.2.3.1 $\wedge$ $\mathcal{A}$ issues a $Corrupt_{SC}$ query on $\hat{A}$," occurs with non–negligible probability. We modify the simulation in the analysis of E.1.1.1 to take as input $X_0, B \in_R \mathcal{G}^*$, and simulate $\hat{B}$'s role as in the analysis of E.2.2 ($\hat{A}$'s role is simulated as in E.1.1.1). If $\mathcal{A}$ succeeds with non–negligible probability, it yields a polynomial time FXCR–1 signature forger which succeeds with non–negligible probability; contradicting Proposition 3.
  - And, if E.2.3.1.2: "E.2.3.1 $\wedge$ $\mathcal{A}$ issues an *EphemeralKeyReveal* query on the test session," occurs with non–negligible probability, we modify the simulation in E.1.1.1 to take as input $A, B \in_R \mathcal{G}^*$, and abort if $\mathcal{A}$ activates a session matching the t–th session at $\hat{A}$. We simulate $\hat{A}$'s role as in E.1.1.2 and $\hat{B}$'s role as in E.2.2. From any valid forgery σ_0, $\mathcal{S}$ outputs $\sigma_0(Y_0^{e_0}B)^{-x_0d_0} = A^{y_0e_0+b}$; and using the oracle replay technique, $\mathcal{S}$ yields an efficient CDH solver, contradicting the GDH assumption.

- And, if $\hat{A}$ follows the second implementation approach, we make $\mathcal{S}$ take as input $A, B \in \mathcal{G}^*$, simulate $\hat{A}$'s role in the same way as that of $\hat{B}$ in E.2.2, and $\hat{B}$'s role as in E.1.1.2, except that when $\mathcal{A}$ activates the t–th session at $\hat{A}$, $\mathcal{S}$ chooses $x_0 \in_R [1, q-1]$ and provides $\mathcal{A}$ with $(\hat{A}, \hat{B}, X_0)$ ($\mathcal{S}$ also aborts if $\mathcal{A}$ activates a session matching the t–th session at $\hat{A}$). If $\mathcal{A}$ succeeds with non–negligible probability, $\mathcal{S}$ outputs with non–negligible probability $A^{y_0 e_0 + b}$, and using the oracle replay technique, $\mathcal{S}$ yields an efficient CDH solver; E.2.3 cannot occur, except with negligible probability.

Reflection Attacks If $\hat{A} = \hat{B}$, E.1 reduces to E.1.1 and E.1.2; in addition E.1.1 reduces to E.1.1.1 and E.1.1.2. The analyses of the events E.1.1.1, E.1.1.2, and E.1.2 hold if $\hat{A} = \hat{A}$; reflections attacks cannot succeed in E.1.

In E.2 (which reduces here to E.2.1 and E.2.2), E.2.1 reduces to E.2.1.2 (the $Corrupt_{SC}$ query is not allowed on $\hat{A}$), if $\mathcal{A}$ succeeds with non–negligible probability, it yields a polynomial time machine $\mathcal{S}$ which on input A outputs with non–negligible probability Y_0 and $(Y_0^{e_0} A)^a$, and $\mathcal{S}$ yields a squaring CDH solver, contradicting the GDH assumption.

Neither E.1 nor E.2 can occur with non–negligible probability, the SMQV protocol is seCK–secure.

3.10 Conclusion

Even if today insufficient, the Bellare–Rogaway model introduces a major approach for the analysis of key agreement protocols. This approach is enhanced and used in the (e)CK models. However, even if considered as advanced, the (e)CK models do not sufficiently capture the intuition of a secure key agreement protocol. Namely, the principle that an attacker should not be able to impersonate a party, unless it knows the party's static key is not guaranteed in the (e)CK models.

In the CK model, the matching sessions definition is inadequate. As shown in section 3.5, some impersonation attacks are not captured in the CK model, and formal CK–security does not necessary yield practical security. In the eCK model, the ephemeral key definition is not careful enough. This allows the design of formally eCK secure protocols which are practically insecure when leakages on ephemeral Diffie–Hellman exponents or intermediate results are considered. We propose the strengthened eCK model, designed with implementation security and efficiency in mind. The seCK model separates from the eCK model, in addition to encompass it. The seCK model provides stronger reveal queries to the adversary, and is not too restrictive. As illustrated with the SMQV protocol, seCK–secure protocols can be built with usual building blocs. When the SMQV protocol is implemented in a distributed environment with an untrusted host machine and a tamper resistant device with ephemeral public keys computed in idle–time, the non–idle time computational effort of the device safely reduce to few non–costly operations.

Future prospects related to the seCK model include the adaptation of the seCK security definition to password based key agreement and group Diffie–Hellman protocols.

There is an underlying identification (implicit or explicit) scheme, in any authenticated key agreement protocol; and there is no known general paradigm for the design a key agreement protocols. We will also be interested in formalizing the security attributes an identification scheme should provide to yield a seCK secure protocol.

Chapter 4

Complementary Analysis of Diffie–Hellman based Protocols

Contents

4.1 Introduction

The Diffie–Hellman protocol [DIF76], remains the basis of many recent key agreement protocols. In this protocol, two parties , say $\hat{A}$ and $\hat{B}$, generate and exchange ephemeral public keys X, Y, and compute the shared secret $Z = Y^x = X^y$; the session key is derived from Z. The protocol is secure against an eavesdropping only attacker; however this is clearly insufficient. The main limitation of the Deffie–Hellman protocol is its lack of authentication, usually illustrated with the well–known man–in-the–attack, wherein the attacker intercepts $\hat{A}$'s message to $\hat{B}$ X, chooses $x' \in [1, q-1]$ and sends $X' = G^{x'}$ to $\hat{B}$; $\hat{B}$'s message to $\hat{A}$ is intercepted and modified in a similar way. In doing so, the attacker impersonates $\hat{A}$ to $\hat{B}$ and conversely.

To prevent the man–in–the–middle attack, the messages exchanged between $\hat{A}$ and $\hat{B}$ can be their static keys, and the shared secret $Z = A^b$; the protocol is termed *static* Diffie–Hellman. However, this variant also is unsatisfactory, as the shared secret does not depend on any random input, the static Diffie–Hellman protocol cannot

achieve the *known session key* security attribute, which is that an adversary which learns some session keys should not be able to compute other session keys.

The above two Diffie–Hellman protocols are widely used for the design of authenticated key agreement protocols. Broadly, the idea is to "mix" the two original DH variants to achieve authentication and randomize session keys. Two main approaches are followed: explicit authentication wherein authentication is achieved using explicit signatures, and implicit signature, wherein authentication is achieved through the sole ability of a party to compute the shared secret. It is usually carried more interest to the later approach, since it generally yields more efficient protocols.

Numerous designs was proposed (a large part of them can be found in the "Key Establishment Protocols Lounge[1]"); unfortunately, in hindsight the rate of secure protocols, particularly when regarded through recent security models, is meager.

In this chapter, we illustrate the two main design approaches for achieving authentication in Diffie–Hellman protocols. We do so using the UM (variant from [NIS07]) and Station–to–Station protocols. After that, we restrict our attention on the design elements of the famous (C, H)MQV protocols. We highlight some shortcomings in the (H)MQV design. On the basis of this analysis we show how impersonation and man in the middle attacks can be performed against the (C, H)MQV protocols, when some session specific information leakages occur [SAR09a, SAR09b]. We define the Full Exponential Challenge Response (FXRC) and Full Dual Exponential Challenge Response (FDCR) signature schemes; and using these schemes we propose the Fully Hashed MQV protocol, which preserves the remarkable performance of the (C, H)MQV protocols and resists the attacks we present [SAR09a, SAR09b]. The FHMQV protocol meets the seCK security definition under the Gap Diffie–Hellman assumption and the Random Oracle model [SAR09a].

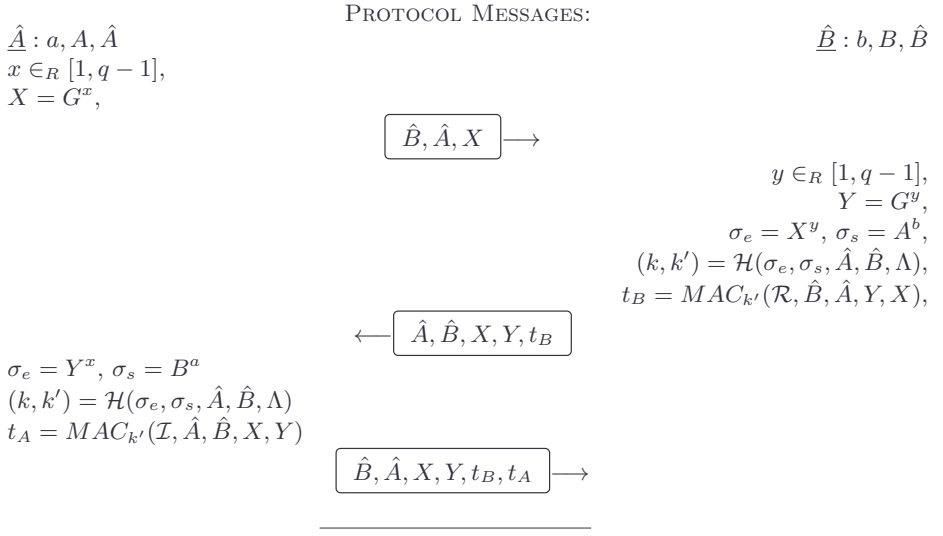
4.2 The Unified Model Protocol

The variant of the UM protocol we discuss in this section is the *dhHybrid1* from the NIST SP800–56A standard [NIS07]. This variant seems to provide more security attributes than the others analyzed in [BLA97a, JEO04]. (In this section, we refer to the NIST dhHybrid1 protocol as UM.) A run of UM, between two parties, say $\hat{A}$ and $\hat{B}$, is as in Protocol 4.1, $\mathcal{H}$ is 2λ-bit hash function, where λ is the desired session key length, MAC is a message authentication code, and Λ designates optional public information that may be used in key derivation.

The design of the UM protocol is rather simple. When the key confirmation stage is ruled out, the core protocol can be viewed as a simultaneous run of the static and non–static Diffie–Hellman protocols. As shown in [MEN08], the UM protocol achieves many important security attributes among which *implicit entity authentication, key secrecy, unknown key share attacks resilience, forward secrecy, known session key security.* However, the protocol fails to be key compromise impersonation resilient. Menezes and Ustaoglu [MEN08, p. 2] argue that key compromise impersonation (KCI) resilience "is arguably not a fundamental security requirement of key agreement". It is also

[1] at https://wiki.isi.qut.edu.au/ProtocolLounge/

Protocol 4.1 UM key exchange



I) The initiator $\hat{A}$ does the following:
   (a) Choose $x \in_R [1, q-1]$, and compute $X = G^x$.
   (b) Send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$.

II) At receipt of $(\hat{B}, \hat{A}, X)$, $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Choose $y \in_R [1, q-1]$, compute $Y = G^y$
   (c) Compute $\sigma_e = X^y$, $\sigma_s = A^b$, and $(k, k') = \mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, \Lambda)$.
   (d) Destroy σ_e, σ_s, and y.
   (e) Compute $t_B = MAC_{k'}(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$.
   (f) Send $(\hat{A}, \hat{B}, X, Y, t_B)$ to $\hat{A}$.

III) At receipt of $(\hat{A}, \hat{B}, X, Y, t_B)$, $\hat{A}$ does the following:
   (a) Verify that $Y \in \mathcal{G}^*$.
   (b) Compute $\sigma_e = Y^x$, $\sigma_s = B^a$, and $(k, k') = \mathcal{H}(\sigma_e, \sigma_s, \hat{A}, \hat{B}, \Lambda)$.
   (c) Destroy σ_e, σ_s, and y.
   (d) Verify that $t_B = MAC_{k'}(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$.
   (e) Compute $t_A = MAC_{k'}(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$, and destroy k'.
   (f) Send $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$ to $\hat{B}$.

IV) At receipt of $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$, $\hat{B}$ verifies that $t_A = MAC_{k'}(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$, and destroys k'.

V) The shared session key is k.

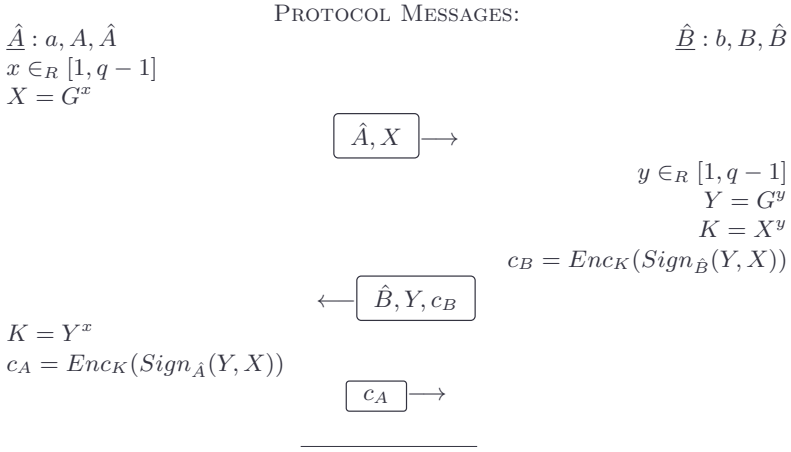
sustainable that, as it is difficult (if not impossible) to guarantee that key compromise can always be immediately detected, and as for non–KCI resilient protocols, when a key comprise occurs, the key owner cannot achieve neither *entity authentication* nor *key secrecy*, which are fundamental goals in authenticated key exchange, KCI resilience is crucial.

Indeed the UM protocol does not only fail to be KCI resilient, it fails also to be resilient to a leakage of σ_s, as an attacker which learns σ_s in some session between $\hat{A}$ and $\hat{B}$ can indefinitely impersonate $\hat{A}$ to $\hat{B}$, and conversely; it can then indefinitely succeed in man–in–the–middle attack between $\hat{A}$ and $\hat{B}$.

4.3 The Station–to–Station Protocol

The Station–to–Station (STS) protocol [DIF92] provides authentication by adding explicit signatures to the non–static Diffie–Hellman protocol. Other variants of the STS protocol exist, wherein authentication is achieved using message authentication or encryption schemes. The original variant which uses an encryption and a signature scheme [DIF92] is given in Protocol 4.2; $Sign$ and Enc are respectively a signature and a public key encryption scheme.

Protocol 4.2 STS using Encryption and Signature Schemes



I) The initiator $\hat{A}$ does the following:
   (a) Choose $x \in_R [1, q-1]$, and compute $X = G^x$.
   (b) Send $(\hat{A}, X)$ to the peer $\hat{B}$.

II) At receipt of $(\hat{A}, X)$, $\hat{B}$ does the following:
   (a) Choose $y \in_R [1, q-1]$, and compute $Y = G^y$.
   (b) Compute $K = X^y$.
   (c) Compute $s_B = Sign_{\hat{B}}(Y, X)$.
   (d) Compute $c_B = Enc_K(s_B)$.
   (e) Send $(\hat{B}, c_b)$ to $\hat{A}$.

III) At receipt of $(\hat{B}, c_b)$, $\hat{A}$ does the following:
   (a) Compute $K = Y^x$.
   (b) Compute $s_B = Dec_K(c_B)$.
   (c) Verify that s_B is a valid signature on (Y, X) with respect to the key B.
   (d) Compute $s_A = Sign_{\hat{A}}(X, Y)$.
   (e) Compute $c_A = Enc_K(s_A)$.
   (f) Send $(\hat{A}, c_A)$ to $\hat{A}$.

IV) $\hat{B}$ does the following:
   (a) Compute $s_A = Dec_K(c_A)$.
   (b) Verify that s_A is a valid signature on (X, Y) with respect to the key A.

V) The shared session key is K.

Notice that in usual practical settings the session key is computed as $K = KDF(Y^x)$, where KDF is a key derivation function. The security arguments from [DIF92] of this variant of STS are rather informal. But, since a session key computation involves only

ephemeral information, an that one cannot make a party complete a session unless the party is provided with a valid fresh[2] signature, it is tenable that the STS protocol provides forward secrecy. In addition, since to impersonate a party, one has to be able to generate a fresh and valid signatures of the party, key compromise impersonation resilience is also achieved.

As shown in [BLA99], when a MAC scheme is used in Protocol 4.2 instead of an encryption scheme, the resulting variant of STS becomes vulnerable to an unknown key share attack. The attacker simply registers a certificate $\hat{E}$ using the target party's public key; it then replaces the target party's certificate with $\hat{E}$ in all messages from the target party. The attack can be prevented in many manners, a simple and non–costly one is using the implicated parties identities in the session key computation ($K = H(Y^x, \hat{A}, \hat{B})$). The STS variants can also can be modified to include the identity of the signer in the MACed and signed data.

4.4 The MQV Protocol

MQV is a key exchange protocol with implicit authentication, based on the Diffie–Hellman protocol. It was proposed by Law, Menezes, Qu, Solinas and Vanstone [LAW03], and is to date one of the most widely standardized key exchange protocols [ANS01a, ANS01b, IEE00, ISO02, NIS03]. When the underlying group is that of the rational points of an elliptic curve E over a finite field, the protocol is denoted ECMQV. If so, we use the following notations: $\Psi = (q, FR, \mathcal{S}, a, b, P, n, h)$ is a domain parameters, where q is the order of the base field $GF(q)$, FR is the field representation, $\mathcal{S}$ is the seed for randomly generated elliptic curves, the coefficients a and $b \in GF(q)$ define the equation of the elliptic curve E over $GF(q)$, P is a point of the curve of order n a large prime, and h is the cofactor of n. We denote by Q_A the static public key of an entity with identity $\hat{A}$, and d_A the corresponding private key. The public keys are supposed to belong to $\langle P \rangle \setminus \{\infty\}$. The affine representation of a point R of the curve E is denoted (x_R, y_R), and $\bar{R}$ is the integer $(\bar{x}_R \bmod 2^l) + 2^l$, where $\bar{x}_R$ is the integer representation of x_R and $l = \lceil (\lfloor \log_2 n \rfloor + 1)/2 \rceil$. The notation MAC still refers a message authentication code scheme and KDF a key derivation function. Validating a public key Q with respect to Ψ consists in [HAN03]

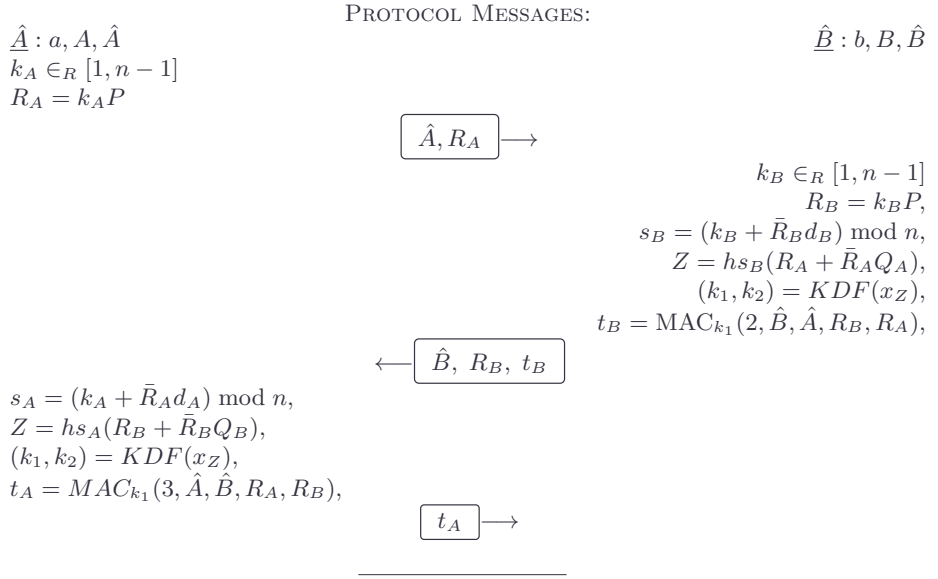
- Verifying that $Q \neq \infty$;
- Verifying that x_Q and y_Q are correctly represented in F_q with respect to FR;
- Verifying that Q satisfies the curve equation defined by a and b;
- And verifying that $nQ = \infty$.

An execution of ECMQV between two entities, say $\hat{A}$ and $\hat{B}$, is as in Protocol 4.3; if any verification fails the execution terminates with failure.

There are also a two and one pass variants of the (EC)MQV protocol. The two–pass variant is obtained by removing the key confirmation step (the tags t_A and t_B and messages in which they are sent) from the tree–pass variant. In the one–pass

[2]Signature freshness is guaranteed by the use of the newly generated ephemeral keys as signed data.

Protocol 4.3 ECMQV key exchange



I) The initiator $\hat{A}$ does the following:
   (a) Choose $k_A \in_R [1, n-1]$.
   (b) Compute $R_A = k_A P$.
   (c) Sends $\hat{A}$, R_A to the peer $\hat{B}$.

II) At receipt of $\hat{A}$, R_A, $\hat{B}$ does the following:
   (a) Validate the ephemeral key R_A.
   (b) Choose $k_B \in_R [1, n-1]$.
   (c) Compute $R_B = k_B P$.
   (d) Compute $s_B = (k_B + \bar{R}_B d_B) \bmod n$, $Z = hs_B(R_A + \bar{R}_A Q_A)$.
   (e) Verify that $Z \neq \infty$.
   (f) Compute $(k_1, k_2) = KDF(x_Z)$, where x_Z is the x–coordinate of Z.
   (g) Compute $t_B = \mathrm{MAC}_{k_1}(2, \hat{B}, \hat{A}, R_B, R_A)$.
   (h) Send $\hat{B}$, R_B, t_B to $\hat{A}$.

III) At receipt of $\hat{B}$, R_B, t_B, $\hat{A}$ does the following:
   (a) Validate the ephemeral key R_B.
   (b) Compute $s_A = (k_A + \bar{R}_A d_A) \bmod n$, $Z = hs_A(R_B + \bar{R}_B Q_B)$.
   (c) Verify that $Z \neq \infty$.
   (d) Compute $(k_1, k_2) = KDF(x_Z)$, where x_Z is the x–coordinate of Z.
   (e) Verify that $t_B = MAC_{k_1}(2, \hat{B}, \hat{A}, R_B, R_A)$.
   (f) Compute $t_A = MAC_{k_1}(3, \hat{A}, \hat{B}, R_A, R_B)$
   (g) Send t_A to $\hat{B}$.

IV) $\hat{B}$ verifies that $t_A = MAC_{k_1}(3, \hat{A}, \hat{B}, R_A, R_B)$.

V) The shared session key is k_2.

variant (motivated by the scenarios in which the responder is off–line) there is no key confirmation, in addition the peer's static public key is used as incoming ephemeral key.

In the ECMQV protocol, the exponent s_A serves as implicit signature for $\hat{A}$'s ephemeral public key R_A, in the sense that only $\hat{A}$ can compute s_A; and this signature

is (indirectly) verified by $\hat{B}$, when using $R_A + \bar{R}_A Q_A = s_A P$ to compute Z. The tags t_A and t_B ensure each entity that the other entity has indeed computed the shared secret Z.

4.4.1 Kunz–Jacques and Pointcheval Security Arguments

Besides the work of Kunz–Jacques and Pointcheval [KUN06], which relies on a somewhat non–standard assumption, recalled hereunder, we are aware of no security reduction for the MQV protocol. In this subsection, we outline the Kunz–Jacques and Pointcheval security arguments and discuss their limitations.

Let f be a function $f : \langle P \rangle \longrightarrow \{0,1\}^l$, the *$f$–Randomized Computational Diffie–Hellman* [KUN06] (f–RCDH) *problem* is: given $R_x = k_x P$ and $R_y = k_y P$, with $R_x, R_y \in_R \langle P \rangle$, find R_s and $R_t \in \langle P \rangle$ such that $R_t = k_x R_s + f(R_s) k_x k_y P = (k_s + f(R_s) k_y) R_x$ where k_s and k_t are respectively $\log_P R_s$ and $\log_P R_t$.

If the considered function f is $f_{\text{MQV}} : R \longrightarrow \bar{R}$, a f–RCDH solver allows to succeed in impersonation attack. Indeed, given a f_{MQV}–RCDH solver, one can impersonate $\hat{B}$ to $\hat{A}$ in any session initiated by $\hat{A}$ (and with intended peer $\hat{B}$), by solving the f–RCDH problem instance with $R_x = R_A + \bar{R}_A Q_A$ and $R_y = Q_B$. One can then provide to $\hat{A}$ R_s as ephemeral key, together with the static one $R_y = Q_B$, and use R_t as secret group element (the Z that $\hat{A}$ derives at step (IIIb) in Algorithm 4.1). And then compute the same session key as $\hat{A}$ does. Notice that this holds for both the two and three–pass variants of the (EC)MQV protocol.

Under the assumption that the f_{MQV}–RCDH problem is intractable (f_{MQV}–RCDH assumption), [KUN06] shows that the MQV protocol is secure in a security model of their own design (inspired from that of Bellare–Rogaway [BEL93a]) which captures *key secrecy*, *mutual authentication* and *weak forward secrecy*.

Moreover, under the assumption that the considered function f is a random oracle, the f–RCDH and the CDH assumptions are equivalent [KUN06]. Hence, under the CDH assumption and the random oracle model, the modified variant of the (EC)MQV protocol, where the secret group element Z is defined as $Z = h(k_A + \bar{H}(R_A) d_A)(R_B + \bar{H}(R_B) Q_B) = h(k_B + \bar{H}(R_B) d_B)(R_A + \bar{H}(R_A) Q_A)$ for some l–bit hash function $\bar{H}$, is secure in the sense of [KUN06].

4.4.2 Limitation of the Security Arguments

Such a variant of the MQV protocol, wherein $f_{\text{MQV}}(R) = \bar{H}(R)$, presents an unpleasant aspect which is not discussed in [KUN06]. Indeed as $\bar{H}$ is a random oracle, Algorithm 4.4 yields a pair (i_0, R_{i_0}) such that $\bar{H}(R_0) = 0$ with probability $\Pr_s \approx 0.63$. As $\Pr\left(H(R) = 0\right) = 1/2^l$, and in Algorithm 4.4, the number of points $R_j = jP$ such that $\bar{H}(R_j) = 0$ is a binomial random variable with parameters $(2^l, 1/2^l)$, the probability that Algorithm 4.4 succeeds is $\Pr_s = 1 - (1 - 1/2^l)^{2^l} \approx 1 - \mathrm{e}^{-1} \approx 0.63$ for l large enough.

Although Algorithm 4.4 is not more efficient than a DLP solver (it requires $\mathcal{O}(2^l)$ point additions and digest computations), it is highly parallelizable; and an attacker which holds such a pair $(i_0, R_{i_0} = i_0 P)$ with $\bar{H}(R_{i_0}) = 0$ can (1) impersonate *any*

Algorithm 4.4 Zero search for l–bit random oracle

Set $R = \infty$;
Set $j = 0$;
while $j \leqslant 2^l$ **do**
    $R = R + P$;
    $j = j + 1$;
    If $\bar{H}(R) = 0$ **return** (j, R);
end while
return "failure";

entity to *any* other entity, and (2) succeed in man–in–the–midlle attack between *any* couple of entities.
To impersonate an entity, say $\hat{A}$, to some other entity $\hat{B}$, the attacker sends Q_A and R_{i_0} to $\hat{B}$; as $H(R_{i_0}) = 0$, it can compute

$$Z = CDH(R_{i_0} + \bar{H}(R_{i_0})Q_A,\ h(R_B + \bar{H}(R_B)Q_B)) = i_0 h(R_B + \bar{H}(R_B)Q_B),$$

where R_B and Q_B are $\hat{B}$'s public keys; and then compute the same session key as $\hat{B}$. And since a man–in–the–middle attack can be performed by simultaneously impersonating $\hat{A}$ to $\hat{B}$ and $\hat{B}$ to $\hat{A}$, it is not difficult to see that given such a pair (j_0, R_0) and attacker can succeed in man–in–the–middle–attack between any pair of parties.

4.4.3 Kaliski's Unknown Key Share Attack

In [KAL01], Kaliski provides an unknown key share attack against the *two–pass* variant of (EC)MQV, if certificate registrations can be performed on–line. The attacker $\mathcal{A}$ makes the session responder, say $\hat{B}$, share a key with a different entity that intended without a knowledge of that; the attack is described in Attack 4.1.

Attack 4.1 Kaliski's unknown key share attack

I) $\hat{A}$ chooses $k_A \in_R [1, n-1]$, computes $R_A = k_A P$ and sends $\hat{A}$, R_A to $\hat{B}$.
II) The attacker $\mathcal{A}$ does the following:
    (a) Intercept $\hat{A}$'s message to $\hat{B}$.
    (b) Choose $u \in_R [1, n-1]$.
    (c) Compute $R_E = S_A - uP$ where $S_A = R_A + \bar{R}_A Q_A$.
    (d) Compute $Q_E = \bar{R}_E^{-1} uP$ ($d_E = \bar{R}_E^{-1} u$).
    (e) Obtain a certificate for the public key Q_E.
    (f) Send to $\hat{E}$, R_E to $\hat{B}$.
III) Both $\hat{A}$ and $\hat{B}$ compute the same key
$K = h(k_A + \bar{R}_A d_A)(R_B + \bar{R}_B Q_B) = h(k_B + \bar{R}_B d_B)(R_A + \bar{R}_A Q_A) = h(k_B + \bar{R}_B d_B)(R_E + \bar{R}_E Q_E)$.

Since the implicit signature

$$S_E = R_E + \bar{R}_E Q_E = S_A - uP + \bar{R}_E \bar{R}_E^{-1} uP = S_A,$$

$\hat{A}$ and $\hat{B}$ share the secret

$$Z = (k_B + \bar{R}_B d_B)S_E = (k_A + \bar{R}_A d_A)S_B,$$

and then derive the same session key, with $\hat{B}$ having the belief that the key is shared with an entity with identity $\hat{E}$. The attacker $\mathcal{A}$ cannot compute K; and, it has to obtain the certificate $\hat{E}$ during the protocol's run. As on–line ACs exist, and obtaining a certificate during a run of (EC)MQV is possible, it is arguable that the two–pass (EC)MQV does not achieve entity authentication.

Fortunately, there exists simple counter–measures against Kaliski's attack, among which adding the protocol with a third–pass (as in three–pass variant), or modifying the key computation to integrate the identities of implicated entities ($K = H(Z, \hat{A}, \hat{B})$ instead of $K = H(Z)$ for instance).

4.5 Complementary Analysis of ECMQV

In this section, we analyze the three–pass (EC)MQV variant; to be concrete, we suppose that the underlying group is that of the rational points of a well chosen elliptic curve.

4.5.1 Points for Impersonation Attack

To make clear the use of the points we introduce next for impersonation attacks, we first formalize our definitions of impersonation and man–in–the–middle attacks.

Definition 14 (Impersonation Attacks)**.** Let Π be a key exchange protocol. An attacker $\mathcal{A}$ is said to succeed in *impersonating* $\hat{A}$ to $\hat{B}$ if:
(a) it succeeds in making $\hat{B}$ run the protocol Π, and derive a session key with the belief that its peer is $\hat{A}$; and
(b) it can efficiently compute the session that $\hat{B}$ derives.
And $\mathcal{A}$ is said to succeed a *man–in–the–middle* attack between $\hat{A}$ and $\hat{B}$ if:
(a) it succeeds in making $\hat{A}$ run the protocol, and derive a session key with the belief that its peer is $\hat{B}$;
(b) it succeeds in making $\hat{B}$ run the protocol, and derive a session key with the belief that its peer is $\hat{A}$;
(c) it can efficiently compute the session keys that $\hat{A}$ and $\hat{B}$ derive.

We term an attack wherein the last conditions (b) and (c) are satisfied, and the condition (a) is modified to "$\mathcal{A}$ succeeds in making $\hat{A}$ run the protocol Π, and derive a session key with the belief that his peer is (an entity with identity) $\hat{A}$" as a *weak man–in–the–middle attack*. Notice that it is not meaningless that an entity establishes a session with another one with the same (identifying) certificate; for instance a hand–held computer may communicate with desktop computer sharing the same (identifying) certificate with it.

In the ECMQV protocol, when a party $\hat{A}$ completes a session with peer $\hat{B}$, the shared secret Z is indeed a combination of the values Q_A, R_A, Q_B, and R_B, with the objective that only the knowledge of one of the couples (d_A, k_A) or (d_B, k_B) permits to compute it. If $(\langle P\rangle \ni R \longrightarrow \bar{R})$ were constant ($\bar{R} = \gamma \in \mathbb{N}$, for all R), it would be easy to impersonate $\hat{A}$ to $\hat{B}$, using $R'_A = kP - \gamma Q_A$, $k \in [1, n-1]$. As $(R \longrightarrow \bar{R})$ is not constant, to succeed in impersonating $\hat{A}$, one has to find a point $R'_A \in \langle P\rangle$ such that $R'_A = \zeta P - \bar{R'}_A Q_A$ where $\zeta \in [1, n-1]$ is known.

Definition 15 (Points for impersonation attack (i–point) [SAR08])**.** Let Ψ be a domain parameters and Q_A a valid public key with respect to Ψ. A point $R'_A \in \langle P\rangle \setminus \{\infty\}$ is said to be an *i–point* for $\hat{A}$, if there exists $\zeta \in [1, n-1]$ such that $R'_A = \zeta P - \bar{R'}_A Q_A$; ζ is said to be the *decomposition*.

Given an i–point for $\hat{A}$ R'_A *and its decomposition* ζ, impersonating $\hat{A}$ to any entity, can be performed as in Attack 4.4 [SAR08], which does not require more computations than an ECMQV execution. Notice that the important aspect is knowing the decomposition of an i–point.

Attack 4.2 Impersonation Attack against ECMQV using a decomposed i–point

Require: An i–point for $\hat{A}$ R'_A and its decomposition ζ.

(1) Send $\hat{A}$, R'_A to the peer $\hat{B}$
(2) Intercept $\hat{B}$'s response $\hat{B}$, R_B, t_B, and do the following:
  (a) Validate the ephemeral key R_B.
  (b) Compute $Z = h\zeta(R_B + \bar{R}_B Q_B)$ and verify that $Z \neq \infty$.
  (c) Compute $(k_1, k_2) = KDF(x_Z)$.
  (d) Verify that $t_B = MAC_{k_1}(2, \hat{B}, \hat{A}, R_B, R'_A)$.
  (e) Compute $t_A = MAC_{k_1}(3, \hat{A}, \hat{B}, R'_A, R_B)$.
  (f) Send t_A to $\hat{B}$.
(3) Use k_2 to communicate with $\hat{B}$ on behalf of $\hat{A}$.

Since R'_A and Q_A are valid public keys and $\zeta \neq 0 \bmod n$, $\hat{B}$'s verifications at steps (IIa) and (IIe) of the ECMQV execution do not fail[3]. Hence $\hat{B}$ sends R_B, t_B at step (IIh). The value of Z that $\hat{B}$ computes at step (IId) (of Protocol 4.3) is

$$Z = hs_B(R'_A + \bar{R'}_A Q_A) = hs_B(\zeta P) = h\zeta(s_B P) = h\zeta(R_B + \bar{R}_B Q_B).$$

This is the value we compute at step (2b) in Attack 4.4. Then the values of k_1 and k_2 we compute at step (2c) are those computed by $\hat{B}$ at step (IIg), and then the test at step (IV) of ECMQV succeeds. Thus the session key we obtain at step (3) is the one that $\hat{B}$ obtains.

Notice also that given an i–point for $\hat{A}$ and its decomposition, and an i–point for $\hat{B}$ and its decomposition, an attacker can indefinitely succeed in man–in–the–middle attack between $\hat{A}$ and $\hat{B}$.

[3]Since the ephemeral private key k_B is chosen at random, the probability that $s_B = 0$ is negligible.

In the following proposition we show the existence of i–points, for any given domain parameters and valid public key.

Proposition 5 (Existence of i–points [SAR08])**.** *Let Ψ be a domain parameters and Q_A a valid public key with respect to Ψ. There exists at least $(n - 2^l - 1)$ i–points for $\hat{A}$.*

Proof. Let $\overline{\langle P \rangle}$ be the image of $\langle P \rangle$ through $(R \longrightarrow \bar{R})$. The cardinal of $\overline{\langle P \rangle}$ is $\leqslant 2^l$. And for every $\bar{Y} \in \overline{\langle P \rangle}$ there is at most one point $R_\infty \in \langle P \rangle$ such that $\bar{R}_\infty = \bar{Y}$ and $R_\infty + \bar{R}_\infty Q_A = \infty$; since the existence of another point $R'_\infty \in \langle P \rangle$, which satisfy $\bar{R}'_\infty = \bar{Y}$ and $R'_\infty + \bar{R}'_\infty Q_A = \infty$, would imply $R_\infty + \bar{Y} Q_A = R'_\infty + \bar{Y} Q_A = \infty$ i.e. $R_\infty = R'_\infty$.

Let $\mathcal{R}_\infty$ be the set of such R_∞ points. The cardinal of $\mathcal{R}_\infty$ is at most 2^l, and every point $R \in \langle P \rangle \setminus \{\mathcal{R}_\infty \cup \{\infty\}\}$ is an i–point for $\hat{A}$. Indeed for a such point R, $R + \bar{R} Q_A \neq \infty$ and since both R and Q_A are in $\langle P \rangle$, there exists some $\zeta \in [1, n-1]$ such that $R + \bar{R} Q_A = \zeta P$, or equivalently $R = \zeta P - \bar{R} Q_A$. The proposition is shown. □

4.5.2 Decomposed i–point Search

This section is about the decomposed i–point search problem (ECIP), which is the following: *given a valid domain parameter Ψ and Q_A, a valid public with respect to Ψ, find $R \in \langle P \rangle$ and $\zeta \in [1, n-1]$ such that $R = \zeta P - \bar{R} Q_A$.*

Any efficient ECDLP solver yields an efficient ECIP solver. The ECIP problem is not harder than the ECDLP problem; we do not know however whether or not the converse is true. We suppose that for $i_u \in_R [0, 2^l - 1]$ and $\zeta_v \in_R [1, n-1]$, the l–least significant bits of the x–coordinate of $R_{u,v} = \zeta_v P - (2^l + i_u) Q_A$ are random.

Naive Search

The naive search consists in computing 2^l points of the form $R_{u,v} = \zeta_v P - (2^l + i_u) Q_A$, with $\zeta_v \in_R [1, n-1]$ and $i_u \in [0, 2^l - 1]$. When the l–least significant bits of $x_{R_{u,v}}$ are supposed random $\Pr(\bar{R}_{u,v} = 2^l + i_u) = 1/2^l$. In these computations, the number of points $R_{u,v}$ such that $\bar{R}_{u,v} = 2^l + i_u$ is a binomial random variable with parameters $(2^l, 1/2^l)$. Hence these computations lead to a decomposed i–point with a success probability $\Pr_s = 1 - (1 - 1/2^l)^{2^l} \approx 1 - e^{-1} \approx 0.63 > 1/2$, for l sufficiently large. When some storage is used (as in Algorithm 4.5), the naive approach requires 2^l point additions plus $2^{\frac{l}{2}+1}$ point multiplications and $\mathcal{O}(2^{\frac{l}{2}+1})$ space complexity. Notice that contrary to the classical parallel collision search (see section 2.5.1), when parallelizing the algorithms 4.1 and 4.5, there is no need that the processors share a common list, the communications between the different processors are only required when a processor finds an i–point, and inform the others.

When testing the naive approach with small values of n we get the results summarized in Table 4.1. The first factor in the *number of examples* column indicates the number of domain parameters[4], the second indicates the number of public keys

[4] The domain parameters are chosen at random: the coefficients a, $b \in_R GF(p)$, the discriminant

Algorithm 4.5 Naive i–point search

Input: P, n, and $Q_A \in \langle P \rangle$.
Output: a decomposed i–point for $\hat{A}$ or "failure".

(1) Compute $2^{\frac{l}{2}}$ couples $(i_u, (2^l + i_u)Q_A)$, $i_u \in_R [0 \cdot \cdot 2^l - 1]$.
(2) Compute $2^{\frac{l}{2}}$ couples $(\zeta_v, \zeta_v P)$, $\zeta_v \in_R [1, n-1]$.
(3) **For** u from 1 to $2^{\frac{l}{2}}$ **do**
  For v from 1 to $2^{\frac{l}{2}}$ **do**
  – Compute $R_{u,v} = \zeta_v P - (2^l + i_u)Q_A$.
  – **If** $\bar{R}_{u,v} = (2^l + i_u)$ **return** $R_{u,v}$, ζ_v.
(4) **return** "failure".

used on each domain parameters, and the third indicates how many times decomposed i–point search was done for each public key.

Table 4.1: Naive i–point search success rate.

size of n	percentage of success	number of examples
15	61.90	10×10×10
20	64.30	10×10×10
25	62.80	10×10×10
30	63.20	10×10×10
35	61.80	5×10×10
40	63.60	5×10×10

Possible Optimizations

The naive search consists simply in building two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ (the first column and the first line in Figure 4.1) of length $2^{l/2}$ and verifying whether or not there is some point $R_{i,j} = R_i + R_j$ with $R_i \in \mathcal{L}_1$ and $R_j \in \mathcal{L}_2$ such that $\bar{R}_{i,j} = i$. The $R_{i,j}$'s can be destroyed, once tested, only the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ need to be stored.

The naive decomposed i–point search is highly parallelizable; when m processors are available, it suffices to provide each processor with an interval to traverse; disjoint intervals covering $\mathcal{L}_1 \times \mathcal{L}_2$ can be conveniently defined.

In what follows, we present the main idea of a still ongoing work on optimized parallelization of decomposed i–point search. As points are represented in affine coordinates in the ECMQV protocol (and then in the i–point definition), and at field level, the most costly operation is inversion, a natural question is about the way in which inversions can be removed at least partly from the decomposed i–point search. We have no general answer for this question, however, for curves defined over binary fields, partly removing inversions in decomposed i–point search is possible.

Suppose the domain parameters' elliptic curve defined over a binary field $GF(2^m)$. The idea is a combination of the López–Dahab affine formulas and the Montgomery

of the corresponding curve is verified to be non-zero and it is verified whether there exists a rational point P of order n satisfying $\lfloor \log_2 n \rfloor = \lfloor \log_2 q \rfloor$.

	P'	$2P'$	$3P'$	$\cdots$	$2^{l/2}P'$
$-(2^l+1)Q$	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$	$\cdots$	$R_{1,2^{l/2}}$
$-(2^l+2)Q$	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$	$\cdots$	$R_{2,2^{l/2}}$
$-(2^l+3)Q$	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$	$\cdots$	$R_{3,2^{l/2}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$-(2^l+2^{l/2})Q$	$R_{2^{l/2},1}$	$R_{2^{l/2},2}$	$R_{2^l,3}$	$\cdots$	$R_{2^{l/2},2^{l/2}}$

Figure 4.1: Naive i–point search illustration

simultaneous inversion algorithm, we recall hereunder. Montgomery's simultaneous inversion algorithm [MON87] (Algorithm 4.6) is based on a generalization of the observation that if $x_i \neq 0$, $\frac{1}{x_1} = x_2\frac{1}{x_1x_2}$ and $\frac{1}{x_2} = x_1\frac{1}{x_1x_2}$. The computation of l inverses reduces to one inversion plus $3(l-1)$ multiplications, and l temporary storage.

Algorithm 4.6 Simultaneous Inversion

Input: $x_1, \cdots, x_l \in GF(q)$.
Ouput: $x_1^{-1}, \cdots, x_l^{-1}$.
(1) Set $c_1 = x_1$
(2) **For** j from 2 to l **do**
$c_j = c_{j-1}x_j$
(3) Compute $u = c_l^{-1}$.
(4) **For** from l downto 2 **do**
(a) Compute $x_j^{-1} = uc_{j-1}$.
(b) Compute $u = ux_j$.
(5) Set $x_1^{-1} = u$.
(6) **Return** $(x_1^{-1}, \cdots, x_l^{-1})$.

Moreover, if $R_1 = (x_1, y_1)$ and $R_2 = (x_2, y_2)$ are two rational points of an elliptic curve defined over a binary field, and $R_2 - R_1 = R = (x, y)$, then the x–coordinate of $R_3 = R_2 + R_1$ is [LOP99]

$$x_3 = x + \frac{x_1}{x_1 + x_2} + \left(\frac{x_1}{x_1 + x_2}\right)^2, \quad \text{if } P_2 \neq P_1, \tag{4.1}$$

$$x_3 = x_1^2 + \frac{b}{x_1^2}, \quad \text{otherwise.} \tag{4.2}$$

Hence, computing the x–coordinate[5] of R_3 requires one field inversion, one squaring,

[5]Notice that in the test in Algorithm 4.5, at step (3), the computation of $\bar{R}_{u,v}$ involves only the x–coordinate of $R_{u,v}$, computing the y–coordinate is then superfluous.

and one field multiplication. To compute the list $\mathcal{L}_2$ containing the couples (j, jP), such that the differences between the jP are known, we choose $\lambda \in_R [1, n-1]$, and set $P' = \lambda P$ and $\mathcal{L}_2 = \{(j, jP') : 1 \leqslant j \leqslant 2^{l/2}\}$. For a randomly chosen public key Q, we then suppose the l–least significant bits of the x–coordinate of $-(2^l + i)Q + jP'$ random.

Let k be a positive integer such that $\mathcal{O}(2^{l/2}k^2)$ temporary storage is "conveniently" feasible, and suppose the x–coordinates of the k^2 points $S_{i,j} = jP' + (2^l + i)(-Q_A)$, with $1 \leqslant i, j \leqslant k$ precomputed. Recall that for a rational point point R, x_R denotes the x–coordinate of R. Let $f(\theta)$ be the polynomial defining the base field $GF(2^m)$, and suppose in addition the $x^{(l)}_{S_{i,j}} = \theta^l x_{S_{i,j}} \bmod f(\theta)$ precomputed for $1 \leqslant l < m$ and $1 \leqslant i, j < k$. For the binary curves used in practice, the NIST curves for instance, such computations require *only few exclusive–or and shift operations* (see [HAN03, pp. 54–56] or [FIP00]).

Now, when traversing the two lists, let R_{i_0,j_0} be the current element, and the elements $R_{i_0-\varepsilon_i,j_0-\varepsilon_j}$, $1 \leqslant \varepsilon_i, \varepsilon_j \leqslant k$ already traversed and temporarily stored. For each elements $R_{i_0+\varepsilon_i,j_0+\varepsilon_j}$, with $1 \leqslant \varepsilon_i, \varepsilon_j < k$, we have

$$\begin{aligned} R_{i_0+\varepsilon_i,j_0+\varepsilon_j} &= R_{i_0,j_0} + S_{\varepsilon_i,\varepsilon_j}, \\ R_{i_0,j_0} - S_{\varepsilon_i,\varepsilon_j} &= R_{i_0-\varepsilon_i,j_0-\varepsilon_j}. \end{aligned}$$

And, from the affine López–Dahab formulas

$$x_{R_{i_0+\varepsilon_i,j_0+\varepsilon_j}} = x_{R_{i_0-\varepsilon_i,j_0-\varepsilon_j}} + \frac{x_{S_{\varepsilon_i,\varepsilon_j}}}{x_{S_{\varepsilon_i,\varepsilon_j}} + x_{R_{i_0,j_0}}} + \Big(\frac{x_{S_{\varepsilon_i,\varepsilon_j}}}{x_{S_{\varepsilon_i,\varepsilon_j}} + x_{R_{i_0,j_0}}}\Big)^2.$$

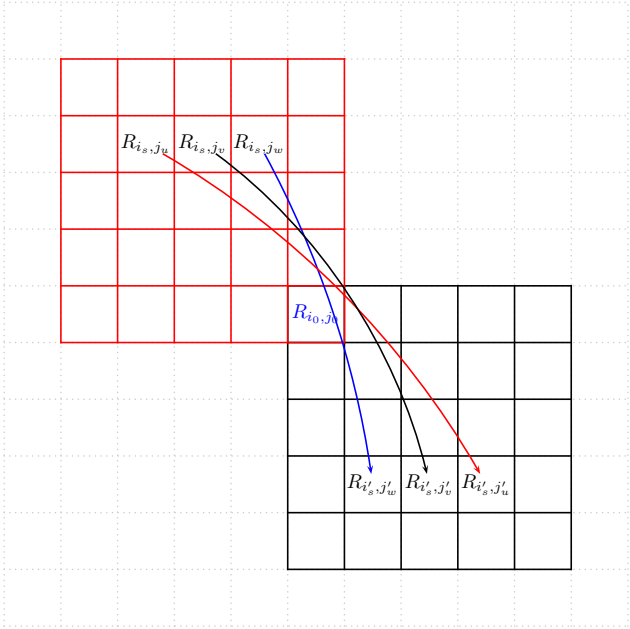


Figure 4.2: López–Dahab coordinates and simultaneous inversion in naive i–point search

As the $x^{(l)}_{S_{i,j}}$ are precomputed, the multiplication in the computation of $x_{R_{i_0+\varepsilon_i,j_0+\varepsilon_j}}$ can be performed using only binary exclusive–or (XOR) operations (roughly $(m-1)/2$ XORs). Recall that if $x, x' \in GF(2^m)$ and if the $x^{(l)} = \theta^l x \bmod f(\theta)$ are known for $1 \leqslant l < m$, then $x'' = x'x \bmod f(\theta)$ can be computed as follows, wherein $\oplus$ denotes the binary exclusive–or (XOR) operation:
(1) If $x'_0 = 1$ then $x'' = x$, else $x'' = 0$.
(2) For t from 1 to $m-1$ do: if $x'_t = 1$ then $x'' = x'' \oplus x^{(l)}$.
(3) Return x''.
Besides the precomputation of the $\theta^l x \bmod f(\theta)$, this is expected to require $(m-1)/2$ XOR operations.

In the computation of each $x_{R_{i_0+\varepsilon_i,j_0+\varepsilon_j}}$, we need an inversion; all the (k^2-1) inversions can be performed simultaneously. Yet, the simultaneous computation of the (k^2-1) inverses $\left(x_{S_{\varepsilon_i,\varepsilon_j}} + x_{R_{i_0,j_0}}\right)^{-1}$, which requires one inversion and $3(k^2-2)$ field multiplications, is still optimizable. Observe that each of the (k^2-1) inversions, involve $x_{R_{i_0,j_0}}$, and as said above, the $x^{(l)}_{S_{i,j}} = \theta^l x_{S_{i,j}} \bmod f(\theta)$ are precomputed. We can then modify the simultaneous inversion algorithm, by first precomputing the $x^{(l)}_{R_{i_0,j_0}} = \theta^l x_{R_{i_0,j_0}} \bmod f(\theta)$ for $1 \leqslant l < m$, such that the multiplications in steps (2) and (4b) of Algorithm 4.6 are performed using only XOR operations. With these modifications, the expected running time of the simultaneous inversion algorithm becomes one inversion and (k^2-1) multiplications (and $(m-1)/2$ field reductions and $2(m-1)(k^2-1)/2$ XOR operations which we consider negligible).

To traverse the two lists, the computations are then performed per bloc (of (k^2-1) elements), and for each non–successful bloc $\mathfrak{b}_{i,j}$ (i.e., a bloc which does not contain an i–point which decomposition is given by an element of $\mathcal{L}_1 \times \mathcal{L}_2$), once $\mathfrak{b}_{i+1,j+1}$ computed, $\mathfrak{b}_{i,j}$ can be destroyed.

The idea is summarized in Algorithm 4.7, further tunning may be possible[6]; notice also that when many processors are available, the precomputations can be performed by an "initiator" processor and passed to the others.

The bloc dimension size k has to be chosen so that $\mathcal{O}(2^{l/2}k^2)$ storage is conveniently feasible. The precomputations require $2^{l/2+1} + k^2$ elliptic curve point additions (the combination of the Montgomery inversion and López–Dahab formula can be used in precomputations for further optimizations). Traversing all the boundary blocs (the blocs $\mathfrak{b}_{1,j}$ and $\mathfrak{b}_{i,1}$) requires $2^{l/2+1}/k$ field inversions + $2^{l/2+1}$ field multiplications, and traversing all the non–boundary blocs requires roughly $\dfrac{(2^{l/2}-k)^2}{k^2}$ field inversions and $(2^{l/2}-k)$ field multiplications.

Using this approach, which requires some precomputations, the number of inversions in the naive search can be reduced by a factor that approximates k^2.

Using Pollard's Rho Algorithm

In this (sub)section, we modify the "simple" rho algorithm (without optimizations from [VAN99, WIE99, GAL00]) to allow decomposed i–point detection [SAR08].

[6] Other ideas, among which incrementing k each time i and j reach the same value to further reduce the number of inversions, or rewriting the simultaneous inversion, are under exploration.

Algorithm 4.7 Optimized i–point search

Input: P, n, and $Q_A \in \langle P \rangle$.
Output: a decomposed i–point for $\hat{A}$ or "failure".

(1) Choose $\lambda \in_R [1, n-1]$, and compute $P' = \lambda P$.
(2) Compute the $2^{\frac{l}{2}}$ couples $(i, -(2^l + i)Q_A)$, for $1 \leqslant i \leqslant 2^{l/2}$.
(3) Compute the $2^{\frac{l}{2}}$ couples (j, jP'), for $1 \leqslant i \leqslant 2^{l/2}$.
(4) Compute the x coordinate of $S_{i,j} = jP' - (2^l + i)Q_A$, for $1 \leqslant i, j \leqslant k$.
(5) For u from 1 to $\dfrac{2^{l/2}}{k-1}$ do
  (a) Check whether the bloc $\mathfrak{b}_{1,u}$ contains a decomposed i–point.
  (b) If so, return the indexes (i_0, j_0) of the i–point.
(6) For u from 1 to $\dfrac{2^{l/2}}{k-1}$ do
  (a) Check whether the bloc $\mathfrak{b}_{u,1}$ contains a decomposed i–point.
  (b) If so, return the indexes (i_0, j_0) of the i–point.
(7) For each bloc $\mathfrak{b}_{i,j}$ with $i, j \leqslant \frac{2^{l/2}}{k-1}$ do
  (a) Check whether the bloc $\mathfrak{b}_{i,j}$ contains a decomposed i–point.
  (b) If so, return the indexes (i_0, j_0) of the i–point.
  (c) If $i, j < \dfrac{2^{l/2}}{k-1}$ then
    - Compute the bloc $\mathfrak{b}_{i+1,j+1}$.
    - Destroy the bloc $\mathfrak{b}_{i,j}$.
  (d) Else, goto step (7).
(8) Return "failure".

To modify the rho method (Algorithm 2.8) for decomposed i–point detection, we need to have $d_2 = -(2^l + \xi) \bmod n$ for some known $\xi < 2^l$.

Definition 16 ([TES01a])**.** Let $r \in \mathbb{N} \setminus \{0\}$, $X_1, \cdots, X_r \in_R \langle P \rangle$, and $g : \langle P \rangle \longrightarrow \{1, \cdots, r\}$ a hash function. A walk $(R_k)_{k \in \mathbb{N}}$ in $\langle P \rangle$ such that $R_0 \in_R \langle P \rangle$, $R_{k+1} = R_k + X_{g(R_k)}$ is said to be an *r–adding*; $\{X_1, \cdots, X_r\}$ is said to be the *supporting set*.

From Teske [TES01a], r–adding walks with an independent hash function and $r \geqslant 16$ in cyclic elliptic curves (sub)groups behave very close to uniformly distributed, with an average period λ and preperiod μ satisfying $\lambda + \mu \leqslant 1.45\sqrt{n}$. Adding a constant term to all elements of a supporting set does not perturb the supporting set's randomness (for $X \in_R \langle P \rangle$ and $C, Y \in \langle P \rangle$, $\Pr(X = Y) = \Pr(X = Y - C) = \Pr(X + C = Y)$). And from [BAI08], fixing the starting value R_0 to some constant does not seem significantly worse than choosing R_0 at random.

Now consider the walk $(R_k)_{k \in \mathbb{N}}$ with starting value $R_0 = X_0 - 2^l Q_A$, $X_0 \in_R \langle P \rangle$, supporting set $\{X_1 - \varepsilon_1 Q_A, \cdots, X_{32} - \varepsilon_{32} Q_A\}$ where $X_k \in_R \langle P \rangle$, and $\varepsilon_k \in_R \{0, 1\}$; with $R_{k+1} = R_k + X_{w(R_k)} - \varepsilon_{w(R_k)} Q_A$, where w is a hash function $w : \langle P \rangle \longrightarrow \{1, \cdots, 32\}$. This walk can be regarded as a "mix" of the following walks.

(a) The r–adding $(R_{j_k})_{k \in \mathbb{N}}$ with starting value $R_{j_0} = X_0$, supporting set $\{X_{j_1}, \cdots, X_{j_r}\}$ (the set of X_i for which $\varepsilon_i = 0$), and with some convenient hash function $w_{(1)} : \langle P \rangle \longrightarrow \{1, \cdots, r_1\}$.

(b) The walk derived from $(R_{j'_k})_{k\in\mathbb{N}}$, by adding all elements of the supporting set of $(R_{j'_k})_{k\in\mathbb{N}}$ with the constant term $-Q_A$; where the starting value and supporting set of $(R_{j'_k})_{k\in\mathbb{N}}$ are $R_{j'_0} = -2^l Q_A$ and $\{X_{j'_1}, \cdots, X_{j'_{32-r}}\}$ (the set of X_i for which $\varepsilon_i = 1$), with some hash function $w_{(2)} : \langle P\rangle \longrightarrow \{1, \cdots, 32-r\}$.

When independent hash functions $w_{(1)}, w_{(2)}$ are used, the walks (R_{j_k}) and $(R_{j'_k})$ are expected to behave close to uniformly distributed. We also expect this for the walk $(R_k)_{k\in\mathbb{N}}$. In the walk $(R_k)_{k\in\mathbb{N}}$, each term R_k can be written $R_k = X - (2^l + \xi)Q_A$ for some known $\xi \leqslant 2^l$ for approximately 2^{l+1} steps. Under the assumptions that the l–least significant bits of the x–coordinate of the R_k are random, and the iterating function is a random one, the probability of detecting an i–point before $1.0308\sqrt{n}$ couples (R_k, R_{2k}) are computed is $\Pr(i) = 1-(1-1/2^l)^{2.0616\sqrt{n}}$; and since $2.0616\sqrt{n} > 2^l$, it follows that $0.63 \approx 1 - e^{-1} \leqslant \Pr(i)$ for n sufficiently large.

When only decomposed i–point detection is considered, the expected number of couples (R_k, R_{2k}) that have to be computed before success is $2^l/2 = 2^{l-1} \approx 1.0308\sqrt{n}/2$. Hence the rho algorithm with modifications to detect i–points is expected approximately twice faster than without modifications. The modified version of the rho algorithm is given in Algorithm 4.8. We get a decomposed i–point for $\hat{A}$ if the return occurs at step (5b) and the private key d_A if it occurs at step (7); and in any of these cases one can succeed in impersonation and weak man–in–the–middle attacks.

Algorithm 4.8 Modified rho algorithm for decomposed i–point detection

Input: P, n , Q_A.
Output: $d_A = \log_P Q_A$, or a decomposed i–point for $\hat{A}$, or "failure".
(1) Choose a partition function $g : \langle P\rangle \longrightarrow \{1, \cdots, L\}$ ($g(R) = j$ if $R \in \mathcal{P}_j$).
(2) **For** j from 1 to L **do**
  (a) Choose $\gamma_j \in_R [0, n-1]$ and $\delta_j \in_R [0 \cdot\cdot 1]$.
  (b) Compute $R^{(j)} = \gamma_j P + \delta_j(-Q_A)$.
(3) Choose $c_1 \in_R [0 \cdot\cdot n-1]$, $d_1 = 2^l$ and compute $R_1 = c_1 P + d_1(-Q_A)$.
(4) Set $R_2 = R_1$, $c_2 = c_1$, $d_2 = d_1$.
(5) **Repeat**
  (a) $j = g(R_1)$, $R_1 = R_1 + R^{(j)}$, $c_1 = c_1 + \gamma_j \bmod n$, and $d_1 = d_1 + \delta_j$;
  (b) **For** i from 1 to 2 **do**
    $j = g(R_2)$, $R_2 = R_2 + R^{(j)}$, $c_2 = c_2 + \gamma_j \bmod n$, $d_2 = d_2 + \delta_j$.
    if $R_2 = d_2$ **return** R_2 and c_2.
  until $R_2 = R_1$.
(6) If $d_1 = d_2$ return "failure".
(7) return $d_A = -(c_1 - c_2)(d_2 - d_1)^{-1} \bmod n$.

Remark 1.
(a) We do not reduce d_2 modulo n since $d_2 \leqslant 2^l$ for approximately 2^{l+1} iterations.
(b) The parallelization technique described in [VAN99] is applicable to this modified version of the rho algorithm.
(c) Algorithm 4.8 applies also if the iterating function is defined over the set of equivalence classes of any group automorphism φ over $\langle P\rangle$ for which $\bar{R} = \overline{\varphi(R)}$ is satis-

fied. In particular, if the iterating function is defined over the set of equivalence classes of the automorphism ($\langle P\rangle \ni R \xrightarrow{\varphi} -R$) [WIE99, GAL00].

Measuring $\mathcal{L}$ = (number of couples (R_i, R_{2i}) computed until success)/$n^{\frac{1}{2}}$, for small values of n, and the average durations of Algorithms 2.8 and 4.8, we get the results in Table 4.2. The number of branches is $L = 32$, $\mathcal{L}_r$ and AD_r (resp. $\mathcal{L}_m$ and AD_m) are respectively the average $\mathcal{L}$ and the average duration (measured in seconds) for the Pollard's rho algorithm (resp. modified version); PM is the percentage of i–points in the results of the modified version. The first factor in the number of examples (NE) column indicates the number of domain parameters, the second indicates the number of instances of the DLP that have been used on each domain parameters, and the third indicates how many times each instance have been used.

Table 4.2: Number of couples (R_i, R_{2i}) computed until success and average duration for the rho algorithm and the modified version — on Magma V2.12_13 on a GNU/Linux Opteron x86-64 4 processors 2390 MHz CPU.

size of n (bits)	$\mathcal{L}_r$	AD_r	$\mathcal{L}_m$	AD_m	PM	NE
20	1.043	0.019	0.462	0.011	72.10	10×10×10
30	1.048	0.766	0.539	0.403	64.80	10×10×10
40	1.038	29.102	0.452	14.363	71.60	10×10×10
50	1.040	995.132	0.489	512.610	69.10	10×10×10
60	1.097	32051.941	0.454	14097.042	63.33	3×2×5

Comparing these values, we see that the modified version is in practice, for sufficiently small values of n, approximately twice faster than the rho algorithm. This conforms to the complexity analysis of the modified rho, and tends to confirms, the expected advantage of the modified algorithm over the rho method.

4.5.3 Exploiting Session Specific Secret Leakages

In this section we show how session specific information leakages can be exploited for impersonation and man–in–the–middle attacks. We consider only leakages the most significant bits; but using the tools from [GOP07] a similar analysis can be performed when considering leakages on middle–part bits.

Impersonation Attack using Session Secret Leakage

Proposition 6 ([SAR08])**.** *Let $\hat{A}$ be a party executing the ECMQV protocol with some peer $\hat{D}$. If an attacker learns the β most significant bits of s_A — defined at step (IIIb) of Protocol 4.1 — then it can indefinitely impersonate $\hat{A}$ to any entity; this requires $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ time complexity and $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ space complexity where $\mu = \lceil \log_2 n \rceil$.*

Remark 2. To meet the two–and–half points multiplications per party performance (or a better), which partly makes the attractiveness of the ECMQV protocol, s_A has to be computed, and the multiplication $(hs_A)(R_D + \bar{R}_D Q_D)$ has to be performed, and

then ephemeral secret exponent (s_A) leakage may occur (through side channel attacks for instance), independently of the ephemeral private key k_A.

Lemma 1 (Shank's Baby Step Giant Step (BSGS) Algorithm [TES01b])**.** *Let $S = sP$ where the β most significant bits of s are known. One can recover s in $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ operations and $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ space complexity.*

Shanks method is deterministic, but requires the storage $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ large integers. Using the Pollard's Kangaroo method [POL78, TES01a], one can compute s with negligible storage, in probabilistic run time $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$.

Lemma 2 ([SAR08])**.** *Let $\hat{A}$ be an entity executing the ECMQV protocol with some peer $\hat{D}$. If an attacker learns the ephemeral secret exponent at $\hat{A}$ (s_A), then it knows an i–point for $\hat{A}$ and its decomposition.*

Proof. Since $\hat{A}$'s static and ephemeral public keys Q_A and R_A are known, it suffices to (re)write $R_A + \bar{R}_A Q_A = s_A P$ i.e. $R_A = s_A P - \bar{R}_A Q_A$. □

Proof of Proposition 6. From Lemma 1, if an attacker learns the β most significant bits of s_A, it can compute s_A in $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ time complexity using $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ space complexity. If the attacker learns s_A, it holds a decomposed i–point for $\hat{A}$ ($R_A + \bar{R}_A Q_A = s_A P$ i.e. $R_A = s_A P - \bar{R}_A Q_A$.); the proposition follows from Attack 4.2. □

From Proposition 6, if an attacker learns (for instance) the half most significant bits of an ephemeral secret exponent at $\hat{A}$, impersonating $\hat{A}$ to any entity requires $\mathcal{O}(n^{\frac{1}{4}})$ operations. Hence Proposition 6 leads to practical attacks when (partial) ephemeral secret exponent leakage occurs, through side channel attacks for instance. This implies also that the ECMQV protocol cannot meet the *loss of information security attribute*, since not only the session in which leakage occured is compromised, but mutual authentication is no more guaranteed for any ECMQV execution implicating $\hat{A}$, while $\hat{A}'s$ static private key is *not* disclosed. Ephemeral secret exponent leakage implies a leakage of the corresponding session key. But, ephemeral secret exponent leakage does not imply neither static leakage, not ephemeral private key leakage. Indeed, one can show that from any algorithm $\mathcal{B}$ which given s_A, R_A and Q_A computes $\hat{A}$'s ephemeral private key k_A or the static one d_A in $C_{\mathcal{B}}$ operations, one can build an algorithm which solves two instances of the discrete logarithm problem in $\langle P \rangle$ in $C_{DLP} + C_{\mathcal{B}}$ operations, where C_{DLP} is the complexity for solving one instance of the DLP in $\langle P \rangle$.

Ephemeral secret exponent leakage implies (but is not equivalent to) *session key reveal*, and does not imply neither *static key reveal* nor *ephemeral key reveal*; while it is not difficult to see that *both ephemeral secret exponent and ephemeral key leakages on the same* session imply the session owner's static key discloser.

Man–in–the–middle Attacks using Session Secret Leakages

We show here that ephemeral secret exponent leakages lead also to man–in–the–middle attacks.

Corollary 1. *Let $\hat{A}$ and $\hat{B}$ be two entities executing the ECMQV protocol with respective peers $\hat{C}$, $\hat{D}$. (i) If an attacker learns the β most significant bits of the ephemeral secret exponent at $\hat{A}$, it can succeed in weak man–in–the–middle attack between $\hat{A}$ and any entity; this requires $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ in time and $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ space complexity. (ii) If in addition, the attacker learns the β most significant bits of the ephemeral secret exponent at $\hat{A}$, then it can indefinitely succeed in man–in–the–middle attack between $\hat{A}$ and $\hat{B}$ this also requires $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ in time and $\mathcal{O}(2^{\frac{\mu-\beta}{2}})$ space complexity.*

If an attacker learns an ephemeral secret exponent at $\hat{A}$, weak man–in–the–middle between $\hat{A}$ and a party, say $\hat{B}'$, is performed by "simultaneously" impersonating $\hat{A}$ to $\hat{B}'$, and $\hat{A}$ to $\hat{A}$, as in Attack 4.3; $s_A^{(l)}$ is the ephemeral secret exponent the attacker learned at $\hat{A}$, and $R_A^{(l)}$ is $\hat{A}$'s outgoing ephemeral public key in the session in which ephemeral secret exponent leakage happened.

Attack 4.3 Weak ECMQV MIM attack using ephemeral secret exponent leakage

(1) Send $\hat{A}, R_A^{(l)}$ to $\hat{B}'$.
(2) Intercept $\hat{B}'$'s response ($\hat{B}'$, $R_{B'}$, $t_{B'}$) and do the following:
  (a) Validate the ephemeral key $R_{B'}$.
  (b) Compute $Z_{B'} = hs_A^{(l)}(R_{B'} + \bar{R}_{B'}Q_{B'})$.
  (c) Verify that $Z_{B'} \neq \infty$.
  (d) Compute $(k_{1_{B'}}, k_{2_{B'}}) = KDF(x_{Z_{B'}})$.
  (e) Verify that $t_{B'} = MAC_{k_{1_{B'}}}(2, \hat{B}', \hat{A}, R_{B'}, R_A^{(l)})$.
  (f) Compute $t'_{A_{B'}} = MAC_{k_{1_{B'}}}(3, \hat{A}, \hat{B}', R_A^{(l)}, R_{B'})$.
  (g) Send $t'_{A_{B'}}$ to $\hat{B}'$.
(3) Send $\hat{A}$, $R_A^{(l)}$ to $\hat{A}$.
(4) At $\hat{A}$'s response ($\hat{A}$, R_A, t_A) do the following:
  (a) Validate the ephemeral key R_A.
  (b) Compute $Z_A = hs_A^{(l)}(R_A + \bar{R}_A Q_A)$.
  (c) Verify that $Z_A \neq \infty$.
  (d) Compute $(k_{1_A}, k_{2_A}) = KDF(x_{Z_A})$.
  (e) Verify that $t_A = MAC_{k_{1_A}}(2, \hat{A}, \hat{A}, R_A, R_A^{(l)})$.
  (f) Compute $t'_{A_A} = MAC_{k_{1_A}}(3, \hat{A}, \hat{A}, R_A^{(l)}, R_A)$.
  (g) Send t'_{A_A} to $\hat{A}$.
(5) Use the key $k_{2_{B'}}$ to communicate with $\hat{B}'$ on behalf of $\hat{A}$.
(6) Use the key k_{2_A} to communicate with $\hat{A}$ on behalf of (a peer with identity) $\hat{A}$.

The man–in–the–middle attack is as in Attack 4.4; $s_A^{(l)}$ and $s_B^{(l)}$ are the ephemeral secret exponents the attacker learns (in previous sessions), and $R_A^{(l)}$ and $R_B^{(l)}$ are respectively $\hat{A}$'s and $\hat{B}$'s outgoing ephemeral public keys in the leaked sessions.

Roughly speaking, Attack 4.4, is simply a simultaneous impersonation $\hat{A}$ to $\hat{B}$, and $\hat{B}$ to $\hat{A}$; this can be performed given an i–point for $\hat{A}$ and an i-point for $\hat{B}$. The session key that $\hat{B}$ derives is k_{2_B} where $(k_{1_B}, k_{2_B}) = KDF(x_{Z_B})$, with

$$Z_B = h(k_B + \bar{R}_B d_B)(R_A^{(l)} + \bar{R}_A^{(l)} Q_A) = hs_A^{(l)}(R_B + \bar{R}_B Q_B).$$

Attack 4.4 MIM attack against ECMQV using ephemeral secret exponent leakages

(1) Send $\hat{A}, R_A^{(l)}$ to $\hat{B}$.
(2) Intercept $\hat{B}$'s response ($\hat{B}$, R_B, t_B) and do the following:
  (a) Validate the ephemeral key R_B.
  (b) Compute $Z_B = hs_A^{(l)}(R_B + \bar{R}_B Q_B)$.
  (c) Verify that $Z_B \neq \infty$.
  (d) Compute $(k_{1_B}, k_{2_B}) = KDF(x_{Z_B})$.
  (e) Verify that $t_B = MAC_{k_{1_B}}(2, \hat{B}, \hat{A}, R_B, R_A^{(l)})$.
  (f) Compute $t_A = MAC_{k_{1_B}}(3, \hat{A}, \hat{B}, R_A^{(l)}, R_B)$.
  (g) Send t_A to $\hat{B}$.
(3) Send $\hat{B}$, $R_B^{(l)}$ to $\hat{A}$.
(4) Intercept $\hat{A}$'s response ($\hat{A}$, R_A, t_A) and do the following:
  (a) Validate R_A.
  (b) Compute $Z_A = hs_B^{(l)}(R_A + \bar{R}_A Q_A)$ and verify that $Z_A \neq \infty$.
  (c) Compute $(k_{1_A}, k_{2_A}) = KDF(x_{Z_A})$ and verify that $t_A = MAC_{k_{1_A}}(2, \hat{A}, \hat{B}, R_A, R_B^{(l)})$.
  (d) Compute $t_B = MAC_{k_{1_A}}(3, \hat{B}, \hat{A}, R_B^{(l)}, R_A)$.
  (e) Send t_B to $\hat{A}$.
(5) Use the key k_{2_B} to communicate with $\hat{B}$ on behalf of $\hat{A}$.
(6) Use the key k_{2_A} to communicate with $\hat{A}$ on behalf of $\hat{B}$.

This is the key the attacker derives for communication with $\hat{B}$ on behalf of $\hat{A}$. Similarly, the session key that $\hat{A}$ derives is k_{2_A} where $(k_{1_A}, k_{2_A}) = KDF(x_{Z_A})$ and $Z_A = hs_B^{(l)}(R_A + \bar{R}_A Q_A)$.

4.6 Complementary Analysis of the HMQV design

In this section we highlight some shortcomings in the HMQV design. The HMQV protocol [KRA05] was designed with the objective to circumvent flaws in the MQV design. Namely, the security of MQV is susceptible to group elements representation, and the protocol cannot be shown secure without further assumptions on the underlying group elements representation. Unfortunately, the HMQV is less secure than stated. Notice that in our description of HMQV, the ephemeral public keys are tested for membership in $\mathcal{G}^*$; while public key validation is voluntarily omitted in [KRA05], the HMQV protocol is known to be insecure if public keys are not correctly validated [MEN06, MEN07].

4.6.1 Exploiting Secret Leakage in the XCR and DCR Schemes

Definition 17 (Exponential Challenge–Response signature [KRA05])**.** Let $\hat{B}$ be an entity with public key $B \in \mathcal{G}^*$, and $\hat{A}$ a verifier. $\hat{B}$'s signature on message a m and challenge X provided by $\hat{A}$ ($X = G^x$, $x \in_R [1, q-1]$ is chosen and kept secret by $\hat{A}$) is $Sign_{\hat{B}}(m, X) = (Y, X^{s_B})$, where $Y = G^y$, $y \in_R [1, q-1]$ is chosen by $\hat{B}$, and

$s_B = y + \bar{H}(Y, m)b$. The verifier $\hat{A}$ accepts a pair (Y, σ_B) provided by $\hat{B}$ as a valid signature if $Y \in \mathcal{G}^*$ and $(YB^e)^x = \sigma_B$, where $e = \bar{H}(Y, m)$.

In this scheme, the information s_B "allows" an attacker to generate valid signatures. Indeed, given the s_B, "corresponding" to some message m and some Y, one can generate a valid signature on any message–challenge pair (m, X_1) (X_1 is a new challenge and the message is unchanged). In the HMQV protocol, the identity of $\hat{B}$ stands for $\hat{A}$'s message to $\hat{B}$, and thus does not change from one session (between $\hat{A}$ and $\hat{B}$) to another; hence (as in MQV) this can be exploited when s_B leakage occurs.

Proposition 7 ([SAR09a]). *Let $\hat{B}$ be an entity, with public key $B \in \mathcal{G}^*$, signing a message–challenge pair (m, X). If an attacker learns the β most significant bits of s_B, then it can generate valid signatures with respect to $\hat{B}$'s public key, on any message–challenge pair (m, X_1) (the message is unchanged); this requires $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ time complexity and $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ space complexity.*

Definition 18 (Dual XCR signature [KRA05]). Let $\hat{A}$ and $\hat{B}$ be two entities with public keys $A, B \in \mathcal{G}^*$; and m_1, m_2 two messages. The Dual XCR (DCR) signature of $\hat{A}$ and $\hat{B}$ on m_1, m_2 is $DSign_{\hat{A},\hat{B}}(m_1, m_2, X, Y) = G^{(x+da)(y+eb)}$, where $X = G^x \in_R \mathcal{G}^*$ and $Y = G^y \in_R \mathcal{G}^*$ are respectively chosen by $\hat{A}$ and $\hat{B}$, $d = \bar{H}(X, m_1)$, and $e = \bar{H}(Y, m_2)$.

In the DCR scheme, once $\hat{A}$ and $\hat{B}$ have exchanged their respective message–challenge pairs (m_1, X) and (m_2, Y), they can both compute the same DCR signature $\sigma_A = (YB^e)^{x+da} = (XA^d)^{y+eb} = \sigma_B$. Notice also that the DCR signature of $\hat{A}$ and $\hat{B}$ on messages m_1, m_2 is an XCR of $\hat{A}$ on the message m_1 and challenge YB^e.

Proposition 8 ([SAR09a]). *Let $\hat{A}$ and $\hat{B}$ be two entities, with public keys $A, B \in \mathcal{G}^*$, signing the messages m_1, m_2, with challenges X, Y. If an attacker learns the β most significant bits of $s_A = x + da$ ($d = \bar{H}(X, m_1)$), then it can compute a valid DCR of $\hat{A}$ and $\hat{B}$ on any message m_2' and challenge Y' from $\hat{B}$; this requires $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ time complexity and $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ space complexity.*

Proof. Since, the DCR signature of $\hat{A}$ and $\hat{B}$ on (m_1, m_2'), is also a XCR signature of $\hat{A}$ on challenge $Y'B^e$ and message m_1, the result follows from Proposition 7. □

As in the MQV protocol, to meet the two–and–half exponentiations per party performance in the DCR scheme, the ephemeral secret exponents have to be computed an the exponentiation $(YB^e)^{s_A}$ performed, and then ephemeral secret exponent leakage may occur independently of the ephemeral private keys.

4.6.2 Exploiting Session Specific Secret Leakages in HMQV

A HMQV key exchange between two parties, say $\hat{A}$ and $\hat{B}$, is as in Protocol 4.9. Roughly speaking, the secret shared between $\hat{A}$ and $\hat{B}$ is a DCR signature with messages fixed to $\hat{A}$ and $\hat{B}$. In [KRA05], Krawczyk presents the XCR scheme as a new variant of the following Schnorr's identification scheme:

Protocol 4.9 HMQV key exchange

PROTOCOL MESSAGES:

$\underline{\hat{A}} : a, A, \hat{A}$	$\underline{\hat{B}} : b, B, \hat{B}$
$x \in_R [1, q-1]$,	
$X = G^x$,	
$\boxed{(\hat{A}, \hat{B}, X)} \longrightarrow$	
	$y \in_R [1, q-1]$,
	$y = G^y$,
$\longleftarrow \boxed{(\hat{B}, \hat{A}, Y)}$	
$s_A = (x + da) \bmod q$,	$s_B = (y + eb) \bmod q$,
$\sigma_A = (YB^e)^{s_A}$	$\sigma_B = (XA^d)^{s_B}$,
$K = H(\sigma_A)$	$K = H(\sigma_B)$.

I) The initiator $\hat{A}$ does the following:
   (a) Choose $x \in_R [1, q-1]$.
   (b) Compute $X = G^x$.
   (c) Send $(\hat{A}, \hat{B}, X)$ to $\hat{B}$.

II) $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Choose $y \in_R [1, q-1]$.
   (c) Compute $Y = G^y$.
   (d) Send $(\hat{B}, \hat{A}, Y)$ to $\hat{A}$.
   (e) Compute $d = \bar{H}(X, \hat{B})$ and $e = \bar{H}(Y, \hat{A})$.
   (f) Compute $s_B = (y + eb) \bmod q$ and $\sigma_B = (XA^d)^{s_B}$.
   (g) Compute $K = H(\sigma_B)$.

III) $\hat{A}$ does the following:
   (a) Verify that $Y \in \mathcal{G}^*$.
   (b) Compute $d = \bar{H}(X, \hat{B})$ and $e = \bar{H}(Y, \hat{A})$.
   (c) Compute $s_A = (x + da) \bmod q$ and $\sigma_A = (YB^e)^{s_A}$.
   (d) Compute $K = H(\sigma_A)$.

IV) The shared session key is K.

(a) The signer $\hat{B}$ chooses $y \in_R [1, q-1]$ and sends $Y = G^y$ to $\hat{A}$.
(b) The verifier $\hat{A}$ chooses $e \in_R [1, q-1]$ and sends e to $\hat{B}$.
(c) $\hat{B}$ computes $s = y + eb$ and sends s to $\hat{A}$.
(d) $\hat{A}$ accepts s as a valid signature if $Y \in \mathcal{G}^*$ and $G^s = YB^e$.

There is however a subtlety: in Schnorr's scheme the random element e, used by $\hat{B}$ when computing s, is *always* provided by the verifier $\hat{A}$; while in the XCR and DCR schemes, when $\hat{A}$'s message m_1 is fixed (to $\hat{B}$ as in all sessions between $\hat{A}$ and $\hat{B}$) the value of e, used when computing s_B, depends only on the ephemeral key Y provided by (the signer) $\hat{B}$. This is precisely what makes replay attacks possible against the XCR and DCR schemes, and the HMQV protocol, when s_A or s_B leakage occurs.

Impersonation Attack using Session Specific Secret Leakage

We show here how ephemeral secret exponent leakage can be used for impersonation attacks. Following the complementary analysis on ECMQV, we define an i–point for HMQV as follows.

Definition 19 (HMQV i–point)**.** Let $\hat{A}$ and $\hat{B}$ be two entities with respective public keys A, $B \in \mathcal{G}^*$. A group element $R \in \mathcal{G}^*$ is said to be a HMQV i–point for $\hat{A}$ to $\hat{B}$ if there exists some $k \in [1, q-1]$ such that $R = G^k A^{-\bar{H}(R,\hat{B})}$; k is said to be the decomposition.

Proposition 9 ([SAR09a])**.** *Let $\mathcal{G} = \langle G \rangle$ be a group with prime order q, $\hat{A}$ and $\hat{B}$ two entities with respective public keys A, $B \in \mathcal{G}^*$. There exists at least $q - (2^l + 1)$ HMQV i–points for $\hat{A}$ to $\hat{B}$.*

As for the MQV protocol, the following proposition links the decomposition of an HMQV i–point to impersonation attack.

Proposition 10 ([SAR09a])**.** *Let $\hat{A}$ and $\hat{B}$ be two entities with respective public keys A, $B \in \mathcal{G}^*$. Given a HMQV i–point for $\hat{A}$ to $\hat{B}$ X' and its decomposition k, one can impersonate $\hat{A}$ to $\hat{B}$ with no more computations than needed by a HMQV execution.*

Attack 4.5 Impersonation attack against HMQV using a decomposed i–point

Require A HMQV i–point for $\hat{A}$ to $\hat{B}$ X' and its decomposition k.

(1) Send $(\hat{A}, \hat{B}, X')$ to $\hat{B}$.
(2) Intercept $(\hat{B}, \hat{A}, Y)$ and do the following:
  (a) Verify that $Y \in \mathcal{G}^*$.
  (b) Compute $\sigma_A = \left(Y B^e\right)^k$ where $e = \bar{H}(Y, \hat{A})$.
  (c) Compute $K = H(\sigma_A)$.
(3) Use K to communicate with $\hat{B}$ on behalf of $\hat{A}$.

Impersonations attacks using ephemeral secret exponent leakages against HMQV can be performed in the same way as against MQV. The impersonation attack against HMQV using ephemeral secret exponent leakage was independently reported by Basin and Cremers [BAS10].

Proposition 11 ([SAR09a])**.** *Let $\hat{A}$ be an entity executing the HMQV protocol with some peer $\hat{B}$. If an attacker learns the β most significant bits of the ephemeral secret exponent at $\hat{A}$, then it can indefinitely impersonate $\hat{A}$ to $\hat{B}$; this requires $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ time complexity and $\mathcal{O}(2^{\frac{|q|-\beta}{2}})$ space complexity.*

Man in the Middle Attack using Session Specific Secret Leakages

If in a HMQV execution between $\hat{A}$ and $\hat{B}$, an attacker learns the ephemeral secret exponent at $\hat{B}$, in addition to the ephemeral secret exponent at $\hat{A}$, it can succeed in a man in the middle attack between $\hat{A}$ and $\hat{B}$. The attack is described as Attack 4.6; $s_A^{(l)}$ and $s_B^{(l)}$ are the ephemeral secret exponents the attacker learns, $X^{(l)}$ and $Y^{(l)}$

are respectively $\hat{A}$ and $\hat{B}$'s outgoing ephemeral public keys in the sessions in which leakages happened. Notice that it is not required that $s_A^{(l)}$ and $s_B^{(l)}$ (partial) leakages happen in matching sessions.

Attack 4.6 MIM attack against HMQV using ephemeral secret exponent leakages

(1) Send $(\hat{A}, \hat{B}, X^{(l)})$ to $\hat{B}$.
(2) Intercept $\hat{B}$'s response to $\hat{A}$ $(\hat{B}, \hat{A}, Y)$ and send $(\hat{B}, \hat{A}, Y^{(l)})$ to $\hat{A}$.
(3) Intercept $\hat{A}$'s response to $\hat{B}$, $(\hat{A}, \hat{B}, X)$.
(4) Compute $\sigma_A = (XA^{d_A})^{s_B^{(l)}}$, where $d_A = H(X, \hat{B})$.
(5) Compute $K_A = H(\sigma_A)$.
(6) Compute $\sigma_B = (YB^{e_B})^{s_A^{(l)}}$, where $e_B = H(Y, \hat{A})$.
(7) Compute $K_B = H(\sigma_B)$.
(8) Use the key K_B to communicate with $\hat{B}$ on behalf of $\hat{A}$.
(9) Use the key K_A to communicate with $\hat{A}$ on behalf of $\hat{B}$.

Roughly, Attack 4.6 is a simultaneously impersonation $\hat{A}$ to $\hat{B}$, and $\hat{B}$ to $\hat{A}$. In $\hat{B}$'s belief, $\hat{A}$ initiates a session with him, with $\hat{A}$'s ephemeral public key being $X^{(l)}$; and in $\hat{A}$'s believe, $\hat{B}$ initiates a session with him, with $\hat{B}$'s ephemeral public key being $Y^{(l)}$. Hence the session key $\hat{A}$ derives is

$$K_A = H((Y^{(l)}B^{e_A})^{x+d_A a}) = H((XA^{d_A})^{s_B^{(l)}}),$$

where $e_A = H(Y^{(l)}, \hat{A})$ and $d_A = H(X, \hat{B})$. This is the K_A we compute in step 5. Similarly, the session key that $\hat{B}$ derives is $K_B = H((YB^{e_B})^{s_A^{(l)}})$ where $e_B = H(Y, \hat{A})$. In Attack 4.6 the communications are initiated by the attacker, but the attack remains possible when communications are initiated by $\hat{A}$ (or $\hat{B}$).

4.7 A New Authenticated Diffie–Hellman Protocol

In this section, we define the Full Exponential Challenge Response (FXCR) and Full Dual exponential Challenge Response (FDCR) schemes [SAR09a], which confine to the minimum the consequences of ephemeral secret exponent leakages; and provide security arguments for these schemes. Using these schemes, we define the Fully Hashed MQV (FHMQV) protocol, which preserves the remarkable performance of the (H)MQV protocols, in addition to ephemeral secret exponent leakage resilience.

4.7.1 Full Exponential Challenge Response Signature scheme

Definition 20 (FXCR signature scheme [SAR09a])**.** Let $\hat{B}$ be an entity with public key $B \in \mathcal{G}^*$, and $\hat{A}$ a verifier. $\hat{B}$'s signature on message m and challenge X provided by $\hat{A}$ ($X = G^x$, $x \in_R [1, q-1]$ is chosen and kept secret by $\hat{A}$) is $FSign_{\hat{B}}(m, X) = (Y, X^{s_B})$, where $Y = G^y$, $y \in_R [1, q-1]$ is chosen by $\hat{B}$, and $s_B = y + \bar{H}(Y, X, m)b$; the verifier $\hat{A}$ accepts a pair (Y, σ_B) as a valid signature if $Y \in \mathcal{G}^*$ and $(YB^{\bar{H}(Y,X,m)})^x = \sigma_B$.

The FXCR scheme delivers all the security attributes of the XCR scheme; in addition the "replay attack" we present in section 4.6 does not hold anymore. Indeed, suppose

an attacker which has learned ${s_B}^{(l)} = y^{(l)} + \bar{H}(Y^{(l)}, X^{(l)}, m)b$. When it is provided with a new challenge X (chosen at random) and the same message m, except with negligible probability $X \neq X^{(l)}$, and then $\bar{H}(Y^{(l)}, X^{(l)}, m) \neq \bar{H}(Y^{(l)}, X, m)$. Hence, to replay $Y^{(l)}$ on the message–challenge pair (m, X), the attacker has to find $s_B = y^{(l)} + \bar{H}(Y^{(l)}, X, m)b$; it is not difficult to see that if it can compute s_B from ${s_B}^{(l)}$, then it can find b from s_B, which is not feasible.

Definition 21 (FXCR signature scheme security [SAR09a])**.** The FXCR scheme is said to be secure in $\mathcal{G}$ if given a public key B, a challenge X_0 ($B, X_0 \in_R \mathcal{G}^*$), and hashing and signing oracles, no adaptive probabilistic polynomial time attacker can output with non–negligible success probability a triple (m_0, Y_0, σ_0) such that:

- (Y_0, σ_0) is a valid signature with respect to the public key B, and the message–challenge pair (m_0, X_0); and
- (Y_0, σ_0) was not obtained from the signing oracle with a query on (m_0, X_0) (freshness).

Using the "oracle replay" technique [POI00], we show that the FXCR scheme is secure in the sense of Definition 21.

Proposition 12 ([SAR09a])**.** *Under the CDH assumption in $\mathcal{G}$ and the RO model, the FXCR signature scheme is secure in the sense of Definition 21.*

Proof. Suppose a probabilistic polynomial time attacker $\mathcal{A}$, which given $B, X_0 \in_R \mathcal{G}^*$ succeeds with non–negligible probability in forging a valid signature, with respect to the public key B and challenge X_0. Using $\mathcal{A}$ we build a polynomial time CDH solver $\mathcal{S}$ which succeeds with non–negligible probability. The solver $\mathcal{S}$ provides $\mathcal{A}$ with random coins, and simulates the digest and signature queries. The interactions between $\mathcal{S}$ and $\mathcal{A}$ are described in Figure 4.3.

Under the RO model, the distribution of simulated signatures is indistinguishable from that of real signatures, except the deviation that happens when $\bar{H}(Y, X, m)$ was queried before. Let Q_h and Q_s be respectively the number of queries that $\mathcal{A}$ asks to the hashing and signing oracles. Since the number of queries to the oracles is less than $(Q_h + Q_s)$, and Y is chosen uniformly at random in $\mathcal{G}$, this deviation happens with probability less than $(Q_h + Q_s)/q$, which is negligible. (As $\mathcal{A}$ is polynomial in $|q|$, both Q_h and Q_s are polynomial in $|q|$.) Hence this simulation is perfect, except with negligible probability. Moreover the probability of producing a valid forgery without querying $\bar{H}(Y_0, X_0, m_0)$ is 2^{-l} (which is negligible). Thus, under this simulation, $\mathcal{A}$ outputs with non–negligible probability a valid and fresh forgery $(Y_0, X_0, m_0, \sigma_0^{(1)})$; we denote $\bar{H}(Y_0, X_0, m_0)$ by $e_0^{(1)}$.

From the forking lemma [POI00], the repeat experiment outputs with non–negligible probability a valid and fresh signature $(Y_0, X_0, m_0, \sigma_0^{(2)})$ with a digest $e_0^{(2)}$, which with probability $1 - 2^{-l}$, is different from $e_0^{(1)}$. Then the computation

$$\left(\frac{\sigma_0^{(1)}}{\sigma_0^{(2)}}\right)^{\left(e_0^{(1)} - e_0^{(2)}\right)^{-1}} = \left(\frac{\left(Y_0 B^{e_0^{(1)}}\right)^{x_0}}{\left(Y_0 B^{e_0^{(2)}}\right)^{x_0}}\right)^{\left(e_0^{(1)} - e_0^{(2)}\right)^{-1}} = B^{x_0}$$

First Run of $\mathcal{A}$:

(a) At $\mathcal{A}$'s digest query on (Y, X, m), $\mathcal{S}$ responds as follows:
- if a value is already assigned to $\bar{H}(Y, X, m)$, $\mathcal{S}$ returns the value of $\bar{H}(Y, X, m)$;
- otherwise, $\mathcal{S}$ responds with $e \in_R \{0,1\}^l$, and sets $\bar{H}(Y, X, m) = e$.

(b) At $\mathcal{A}$'s signature query on (m, X), $\mathcal{S}$ responds as follows:
- $\mathcal{S}$ chooses $s_B \in_R [1, q-1]$, $e \in_R \{0,1\}^l$, sets $Y = G^{s_B} B^{-e}$ and $\bar{H}(Y, X, m) = e$. If $\bar{H}(Y, X, m)$ was previously defined, $\mathcal{S}$ aborts;
- $\mathcal{S}$ responds with (Y, X^{s_B}, s_B) (notice that the forger is given s_B in addition to X^{s_B}).

(c) At $\mathcal{A}$'s halt, $\mathcal{S}$ verifies that $\mathcal{A}$'s output $(Y_0, X_0, m_0, \sigma_0)$ (if any) satisfies the following conditions. If not, $\mathcal{S}$ aborts.
- $Y_0 \in \mathcal{G}^*$ and $\bar{H}(Y_0, X_0, m_0)$ was queried from $\bar{H}$.
- The signature (Y_0, σ_0) was not returned by $\hat{B}$ on query (m_0, X_0).

Repeat: $\mathcal{S}$ executes a new run of $\mathcal{A}$, using the same input and coins; and answering to all digest queries before $\bar{H}(Y_0, X_0, m_0)$ with the same values as in the previous run. The new query of $\bar{H}(Y_0, X_0, m_0)$ and subsequent queries to $\bar{H}$ are answered with new random values.

Output: If $\mathcal{A}$ outputs a second signature on $(Y_0, X_0, m_0, \sigma_0)$ satisfying conditions of step (c), with a hash value $\bar{H}(Y_0, X_0, m_0)_2 = e_0^{(2)} \neq e_0^{(1)} = \bar{H}(Y_0, X_0, m_0)_1$, then $\mathcal{S}$ outputs $\left(\sigma_0^{(1)} / \sigma_0^{(2)}\right)^{(e_0^{(1)} - e_0^{(2)})^{-1}}$ as a guess for $CDH(B, X_0)$.

Figure 4.3: Building a CDH solver from a FXCR forger

yields $CDH(B, X_0)$ with non–negligible probability. Recall that such a polynomial time CDH solver, succeeding with non–negligible probability, can be transformed into an efficient CDH solver [MAU96]. □

4.7.2 Full Dual Exponential Challenge Response Signature scheme

Definition 22 (FDCR signature scheme [SAR09a])**.** Let $\hat{A}$ and $\hat{B}$ be two entities with public keys $A, B \in \mathcal{G}^*$, and m_1, m_2 two messages. The FDCR signature of $\hat{A}$ and $\hat{B}$ on messages m_1, m_2 is $FDSign_{\hat{A},\hat{B}}(m_1, m_2, X, Y) = G^{(x+da)(y+eb)} = (XA^d)^{y+eb} = (YB^e)^{x+da}$, where $X = G^x \in_R \mathcal{G}^*$ is chosen by $\hat{A}$ (resp. $Y = G^y \in_R \mathcal{G}^*$ is chosen by $\hat{B}$), $d = \bar{H}(X, Y, m_1, m_2)$, and $e = \bar{H}(Y, X, m_1, m_2)$.

In the FDCR scheme, as in the DCR scheme, once $\hat{A}$ and $\hat{B}$ have provided their respective message–challenge pairs, they can both compute the same signature. However, contrary to the DCR and XCR schemes, the FDCR signature of $\hat{A}$ and $\hat{B}$ on messages m_1, m_2 and challenges X, Y, is *not* a FXCR signature of $\hat{A}$ on m_1 and YB^e.

Definition 23 (FDCR scheme Security [SAR09a])**.** The FDCR scheme is said to be secure in $\mathcal{G}$, if given a, A, B, X_0, and a message m_{1_0}, and hashing and signing oracles, no adaptive probabilistic polynomial time attacker, can output with non–negligible success probability a triple (m_{2_0}, Y_0, σ_0) such that:
- $(m_{1_0}, m_{2_0}, X_0, Y_0, \sigma_0)$ is a valid FDCR signature with respect to the keys A and B.

- (Y_0, σ_0) was not obtained from the signing oracle with a query on (m'_1, X') such that $X_0 = X'$ and $(m'_1, m'_2) = (m_{1_0}, m_{2_0})$, where m'_2 is the message returned at signature query on (m'_1, X'), and (m_{1_0}, m_{2_0}) denotes the concatenation of m_{1_0} and m_{2_0} (freshness).

Remark 3. Since we suppose that if $\hat{A} \neq \hat{A}'$, no substring of $\hat{A}$ equals $\hat{A}'$ (and conversely), if $\hat{A} \neq \hat{A}'$ or $\hat{B} \neq \hat{B}'$ then $(\hat{A}, \hat{B})$ cannot equal $(\hat{A}', \hat{B}')$.

Proposition 13 ([SAR09a]). *Under the CDH assumption in $\mathcal{G}$ and the RO model, the FDCR signature scheme is secure in the sense of Definition 23.*

Proof. Suppose an attacker $\mathcal{A}$, which given a, A, B, X_0, m_{1_0} (with $A \neq B$) outputs with non–negligible success probability a valid and fresh signature forgery (m_{2_0}, Y_0, σ_0). Using $\mathcal{A}$ we build a polynomial time FXCR forger, which succeeds with non–negligible probability. The forger $\mathcal{S}$ provides $\mathcal{A}$ with random coins, a, A, B, X_0, m_{1_0}, and simulates the role of $\hat{B}$ as follows.

(1) At $\mathcal{A}$'s digest query on (X, Y, m_1, m_2), $\mathcal{S}$ responds as follows:
- if a value is already assigned to $\bar{H}(X, Y, m_1, m_2)$, $\mathcal{S}$ returns the value of $\bar{H}(X, Y, m_1, m_2)$;
- otherwise $\mathcal{S}$ responds with $d \in_R \{0,1\}^l$, and sets $\bar{H}(X, Y, m_1, m_2) = d$.

(2) At signature query on (m_1, X), $\mathcal{S}$ responds as follows.
- $\mathcal{S}$ chooses $m_2 \in \{0,1\}^*$, $s_B \in_R [1, q-1]$, $d, e \in_R \{0,1\}^l$, computes $Y = G^{s_B} B^{-e}$, and sets $\bar{H}(X, Y, m_1, m_2) = d$, $\bar{H}(Y, X, m_1, m_2) = e$; if $\bar{H}(X, Y, m_1, m_2)$ or $\bar{H}(Y, X, m_1, m_2)$ was defined in a previous query, $\mathcal{S}$ aborts.
- $\mathcal{S}$ provides $\mathcal{A}$ with the signature $((m_2, Y), (XA^d)^{s_B}, s_B)$ (s_B is returned, with the signature).

Figure 4.4: Building a FXCR forger from a FDCR forger

The simulation of $\hat{B}$'s role is perfect, except with negligible probability. The deviation happens when the same message–challenge pair (m_2, Y) is chosen twice in two signature queries on the same pair (m_1, X). Since Y is chosen uniformly at random in $\mathcal{G}$, this happens with negligible probability. Then if $\mathcal{A}$ succeeds with non–negligible probability in forging a valid and fresh signature σ_0, it succeeds also with non–negligible probability under this simulation. And since $\mathcal{S}$ knows a, using $\mathcal{A}$, it produces with non–negligible success probability

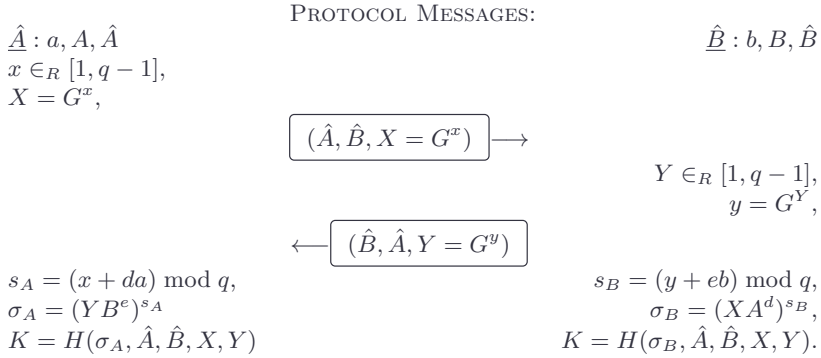
$$\sigma_0 (Y_0 B^e)^{-da} = (Y_0 B^e)^{x_0 + da} (Y_0 B^e)^{-da} = (Y_0 B^e)^{x_0} = {X_0}^{y_0 + eb}.$$

This is valid FXCR forgery on message (m_{1_0}, m_{2_0}) (the concatenation of m_{1_0} and m_{2_0}) and challenge X_0 with respect to the public key B; contradicting Proposition 12. □

4.7.3 The Fully Hashed MQV Protocol.

We can now derive the Fully Hashed MQV (FHMQV) protocol, which provides all security attributes of the (H)MQV protocol, in addition to ephemeral secret exponent

Protocol 4.10 FHMQV key exchange



I) The initiator $\hat{A}$ does the following:
   (a) Choose $x \in_R [1, q-1]$.
   (b) Compute $X = G^x$.
   (c) Send $(\hat{A}, \hat{B}, X)$ to $\hat{B}$.

II) $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Choose $y \in_R [1, q-1]$.
   (c) Compute $Y = G^y$.
   (d) Send $(\hat{B}, \hat{A}, Y)$ to $\hat{A}$.
   (e) Compute $d = \bar{H}(X, Y, \hat{A}, \hat{B})$, $e = \bar{H}(Y, X, \hat{A}, \hat{B})$.
   (f) Compute $s_B = (y + eb) \bmod q$ and $\sigma_B = (XA^d)^{s_B}$.
   (g) Compute $K = H(\sigma_B, \hat{A}, \hat{B}, X, Y)$.

III) $\hat{A}$ does the following:
   (a) Verify that $Y \in \mathcal{G}^*$.
   (b) Compute $d = \bar{H}(X, Y, \hat{A}, \hat{B})$, $e = \bar{H}(Y, X, \hat{A}, \hat{B})$
   (c) Compute $s_A = (x + da) \bmod q$ and $\sigma_A = (YB^e)^{s_A}$.
   (d) Compute $K = H(\sigma_A, \hat{A}, \hat{B}, X, Y)$.

IV) The shared session key is K.

leakage resilience. While using the same overall design as the (H)MQV protocols, FHMQV provides security attributes that are lacking in (H)MQV protocols. Ephemeral secret exponent leakage resilience is provided in FHMQV, while not in the (C,H)MQV protocols.

Theorem 4. *Let $s_A = x + da$ and $\sigma = (YB^e)^{s_A}$, where $d = \bar{H}(X, Y, \hat{A}, \hat{B})$ and $e = \bar{H}(Y, X, \hat{A}, \hat{B})$, be the intermediate results in a session at $\hat{A}$ with peer $\hat{B}$. Under the GDH assumption in $\mathcal{G}$, and the RO model, the FHMQV protocol is seCK–secure.*

The proof of Theorem 4 is similar to that of Theorem 3 (see section 3.9.1), so we omit it. Instead, we summarize the most important differences between the HMQV and FHMQV protocols.

Differences between the FHMQV and HMQV Designs

Notice that, we consider the HMQV variant wherein ephemeral keys are tested for membership in $\mathcal{G}^*$, as if not, HMQV is already known to be insecure [MEN07, MEN06].

Building blocks and adversary model. The design of FHMQV relies on the FXCR and FDCR signature schemes. While in the XCR scheme as in the FXCR scheme, both ephemeral secret exponent and ephemeral key leakages in the same session imply a discloser of the session owner's static private key. In the FXCR scheme, an adversary which has learned an ephemeral secret exponent at a party is unable to forge the party's signature. The seCK model allows ephemeral secret exponent leakage. Better, in the FXCR and FDCR security arguments, when the attacker issues a signature query, it is also provided with the signature's ephemeral secret exponent. The impersonation and man in the middle attacks we presented in section 4.6 do not hold against the FHMQV protocol. An immediate consequence of this security attribute is that, as for the SMQV protocol, when implementing FHMQV in a distributed environment with a computationally limited tamper–resistant device together with an untrusted host machine (see Figure 4.5), the ephemeral keys can be computed in the device in idle–time, while the exponentiation $\sigma = (YB^e)^{s_A} = (XA^d)^{s_B}$ is computed on the host machine.

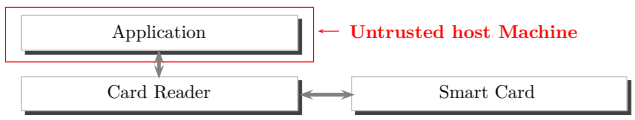


Figure 4.5: Particularly suited FHMQV implementation environment

As for SMQV, the non–idle computational effort of the device reduces to one digest computation, one integer addition and one multiplication.

Key replication attacks resilience. At session key derivation in FHMQV, ephemeral keys and peers identities are hashed with the session's FDCR signature ($K = H(\sigma, \hat{A}, \hat{B}, X, Y)$). Since non matching sessions cannot have (except with negligible probability) the same ephemeral keys, and non matching digest queries cannot have (except with negligible probability) the same digest value, the analysis of key replication attacks is immediate for the FHMQV protocol.

Computational asumptions. The security of the HMQV protocol relies on the GDH, the Knowledge of Exponent Assumption (KEA1) [BEL04] and the RO model. For the FHMQV protocol the (KEA1) assumption is not needed; we only require the RO model and the GDH assumptions.

The FHMQV–C Protocol

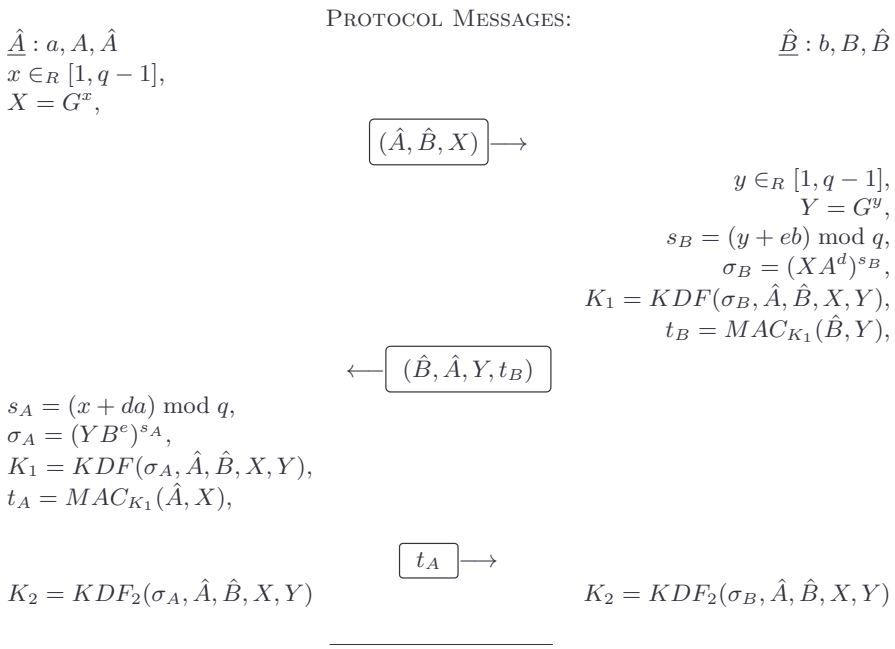
As shown in [KRA05], no implicitly authenticated two–message protocol such as ours can achieve the forward secrecy security attribute; key confirmation security attribute

(for both peers) cannot be achieved also. Nevertheless these security attributes may be desirable; the FHMQV protocol can be added with a third message, yielding the FHMQV–C protocol, we describe in Protocol 4.11; KDF_1 and KDF_2 are key derivation functions, and MAC a message authentication code. If any verification fails, the execution aborts.
When a party, say $\hat{A}$, completes a FHMQV–C session with some honest peer $\hat{B}$, and with incoming ephemeral key Y, it is guaranteed that Y was chosen and authenticated by $\hat{B}$, and that $\hat{B}$ can compute the session key it derives. The FHMQV–C protocol provides also perfect forward secrecy, the compromise of $\hat{A}$'s static private key, does not compromise the session keys established in previous runs. This can be shown when the analysis of FHMQV is completed with the *session–key expiration* notion [CAN01].

4.8 Conclusion

We introduced new points, i–points, for impersonation attacks against the (C, H)MQV(–C) protocols, and showed their existence for any valid domain parameters. We explored the search of these points. The method we propose for decomposed i–point search is expected to be twice faster than the Pollard's rho algorithm. We proposed a complementary analysis of the Exponential Challenge Response and Dual Exponential Challenge Response signature schemes, which are the building blocks of the (H)MQV protocols. On the basis of this analysis, we showed how impersonation and man in the middle attacks can be performed against the (H)MQV protocols, when some session specific information leakages occur. We proposed the Full Exponential Challenge Response (FXCR) and Full Dual Exponential Challenge Response (FDCR) signature schemes, with security arguments. Using these schemes, we defined the Fully Hashed MQV (FHMQV) protocol, which preserves the efficiency of the (H)MQV protocols, and meets the seCK security definition. As for SMQV, FHMQV is particularly suited for distributed environments wherein a tamper resistant device is used with an untrusted machine. For future work, we will be interested in further investigations parallelizations and optimizations the decomposed i–point search. Decomposed i–point search is not harder than the (EC)DLP, however we do not know whether the converse is true or not; we will also be interested in investigating this question.

Protocol 4.11 FHMQV–C key exchange



I) The initiator $\hat{A}$ does the following
   (a) Choose $x \in_R [1, q-1]$.
   (b) Computes $X = G^x$.
   (c) Send $(\hat{A}, \hat{B}, X)$ to $\hat{B}$.

II) At receipt of $(\hat{A}, \hat{B}, X)$ $\hat{B}$ does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Choose $y \in_R [1, q-1]$.
   (c) Compute $Y = G^y$.
   (d) Compute $d = \bar{H}(X, Y, \hat{A}, \hat{B})$ and $e = \bar{H}(Y, X, \hat{A}, \hat{B})$.
   (e) Compute $s_B = (y + eb) \bmod q$ and $\sigma_B = (XA^d)^{s_B}$.
   (f) Compute $K_1 = KDF_1(\sigma_B, \hat{A}, \hat{B}, X, Y)$ and $t_B = MAC_{K_1}(\hat{B}, Y)$.
   (g) Send $(\hat{B}, \hat{A}, Y, t_B)$ to $\hat{A}$.

III) At receipt of $(\hat{B}, \hat{A}, Y, t_B)$, $\hat{A}$ does the following:
   (a) Verify that $Y \in \mathcal{G}^*$.
   (b) Compute $d = \bar{H}(X, Y, \hat{A}, \hat{B})$ and $e = \bar{H}(Y, X, \hat{A}, \hat{B})$.
   (c) Compute $s_A = (x + da) \bmod q$ and $\sigma_A = (YB^e)^{s_A}$.
   (d) Compute $K_1 = KDF(\sigma_A, \hat{A}, \hat{B}, X, Y)$.
   (e) Verify that $t_B = MAC_{K_1}(\hat{B}, Y)$.
   (f) Compute $t_A = MAC_{K_1}(\hat{A}, X)$.
   (g) Send t_A to $\hat{B}$.
   (h) Compute $K_2 = KDF_2(\sigma_B, \hat{A}, \hat{B}, X, Y)$

IV) At receipt of t_A, $\hat{B}$ does the following:
   (a) Verify that $t_A = MAC_{K_1}(\hat{A}, X)$.
   (b) Compute$K_2 = KDF_2(\sigma_B, \hat{A}, \hat{B}, X, Y)$.

V) The shared session key is K_2.

Chapter 5

Implementations of the PKCS #11 Standard

Contents

5.1 Introduction

Cryptographic schemes are widely used in distributed systems. In such systems, the different components may be developed by different teams which may be from different companies. Even if the basic cryptographic schemes are standardized, there is a need for a standard way of communication between the components. PKCS #11 is a communication interface between applications and cryptographic devices. The standard is designed with smart–cards in mind; it is widely adopted in industry, and deployed in many security tokens: the *Globull* from Bull [BUL], the *Smart Enterprise Guardian* from Gemalto [GEM], to mention only a few. Some software open source implementations, among which the NSS–PKCS #11 [MOZ] (which significantly deviates from the standard specification) and the Lite Security Module PKCS #11 (a deamon based implementation [MER09]), are also deployed.

Even if widely deployed, the PKCS #11 standard may be insecure, in particular, when implemented without a strict security policy. Indeed, keys may have conflicting attributes, which make them vulnerable to an attacker which gains access to an authenticated session; in addition, the standard specification gives only few advices on this point. In this chapter, we discuss some of the limitations of the PKCS #11 standard specification, and the ways to circumvent them in practical implementations. We also discuss some of the design choices and technicalities of two PKCS #11 implementations at Netheos. Because of the commercial nature of the products in which the implementations are used, the discussion is voluntarily limited.

In the next section, we discuss the context of the work; in section 5.3, we give an overview of the PKCS #11 specification. Section 5.4 is about the (in)security in the standard specification and implementations, we discuss sensitive keys export, when security conflicting security attributes are allowed; we also discuss key space reduction, and key wrapping based fault attacks. Section 5.5 deals with few of the technicalities in our implementations, concerning the Everbee "Smart Mobile Key" (SMK) and the Netheos' Reconfigurable Cryptographic Platform (RCP).

5.2 Context of the Work

Our work on PKCS (Public–Key Cryptography Standards) #11 is initially motivated by the aim of the Netheos company [NET] to develop secure mobile peripherals, together with tools for a fine management of these peripherals. The whole project is termed eKeynox™[1]. A secure mobile peripheral provides an environment including a secure storage, and some executable embedded applications (a customized Firefox, some applications developed by Netheos, the Sumatra PDF viewer, etc.) to fulfill usual needs.

For the management of the devices, a *registration directory* is designed, to verify the identity of a certificate issuer, together with a *publication center* for a centralized distribution of the software and firmware updates. Different types of *actors* exist, among which the *"help–desk operator",* who can revoke a particular device (by revoking the corresponding owner's certificate, and removing the device from the list of the devices that can connect to the *publication center*), or create a new certificate (using the *registration directory*) for device activation. A *user* is a daily owner of a device. Some intermediate roles exist between these two extremes; broadly, these intermediate roles are partial or total delegations of the *help–desk operator*'s role on a subset of the set of all users.

In the eKeynox context, as many applications use the same cryptographic objects and functions, there is clearly a need for a standard way of communication between the applications and the module providing cryptographic services. Moreover, as many of the applications in the environment are not developed by Netheos, the communication interface provided by the cryptographic services provider has to operate for all applications in the environment. The most popular cryptographic communication interfaces seem to be the IBM Common Cryptographic Architecture (CCA) [IBM08], and the RSA Public Key Cryptography Standards #11 (PKCS #11) [RSA04]. In the eKeynox project, the main application software is developed in Java. And, a PKCS #11 module can be plugged into the SUN Java Cryptographic Architecture [SUN06] (the high–level cryptographic API — which now encompasses the Java Cryptographic Extension), in Firefox, or in Thunderbird, with no development effort; this is not the case for the IBM CCA. In the eKeynox project, a PKCS #11 module seems more adequate than a CCA one. Our contributions in the eKeynox project included

(1) the architecture design of the PKCS #11 module;

(2) the specification of the embedded system's API regarding the PKCS #11 module;

[1] eKeynox is a trademark registered by Netheos.

(3) the test of this API;
(4) the implementation and test of the submodules, except for those in the embedded system;
(5) the integration and test of the involved submodules.

Although the eKeynox solution is commercialized with many devices, the discussion is restricted to the "Smart Mobile Key" (SMK) from Everbee [EVE].

Another project which has motivated this work is Netheos' Reconfigurable Cryptographic Platform (RCP) Project. The goal of Netheos, with this project, is to develop sufficiently generic and modular VHDL and C source codes to permit rapid prototyping of hardware security modules (HSMs) or cryptographic accelerators, depending on customers needs; the target market is mainly the corporate and bank servers segment. Cryptographic devices providing a PKCS #11 interface are widely used in the design of Public Key Infrastructures (PKIs), Virtual Private Networks (VPNs), and so on; providing this interface in the RCP project make the integration of an RCP in such tools convenient. Our contributions in the RCP project included

(1) the co–design of the system;
(2) the software submodules co–development (including the PKCS #11 interface), and
(3) the co–development of the modules running in the embedded processor.

5.3 An Overview of the PKCS #11 Specification

The PKCS #11 standard API, also termed Cryptoki, was proposed by RSA Laboratories [RSAL], in an open cooperation with industries and academias. The goal of the standard is to provide a standard way of communication between applications and portable cryptographic devices, to abstract the devices specificities and allow cryptographic resources sharing (many applications using simultaneously many cryptographic devices).

5.3.1 PKCS #11 Terminology

In the PKCS #11 terminology, a *token* is a device which stores objects, and performs cryptographic treatments using the objects it stores. A *slot* is a logical view of a token reader; when tokens are removable, a slot may not contain a token. In practice, tokens behave as cryptographic auxiliaries, storing cryptographic keys and implementing a variety of cryptographic mechanisms, for applications running on a host machine. For smart–card based implementations, for instance, a token corresponds to the card, and a slot to the card–reader. The slot and token notions are however logical, and the implementations may be software. Also, the perimeter of the cryptographic mechanisms that should be supported by a token is not specified by the standard, this follows from the needs in the context in which the token is intended to be used.

Objects and users. The objects contained in a token can have one of the following "types" (see Table 5.1 for PKCS #11 prefixes). The CKO_HW_FEATURE is the class of the hardware features that may exist in a token: real–time clock, monotonic counters, and so on; these objects are usually "read–only". An object with type

CKO_MECHANISM provides information about a cryptographic mechanism supported by a token. The *storage* objects, which are usually the most commonly used, include data objects, keys objects, certificates objects, and domain parameters. The objects hierarchy is summarized in Figure 5.1 [RSA04, p. 62].

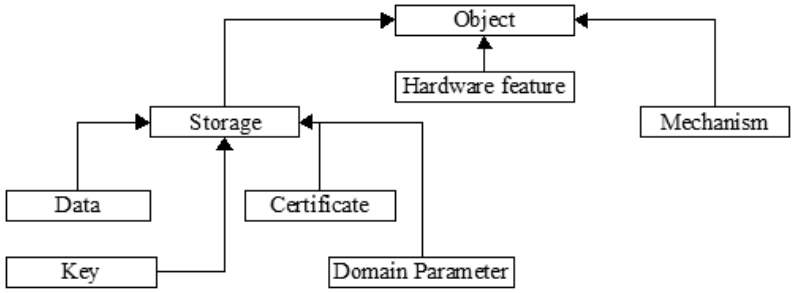


Figure 5.1: Objects Hierarchy in PKCS #11

Table 5.1: PKCS #11 naming prefixes.

Prefixes	**Meanings**
C_	function prefix
CK_	general constant or low level data type
CKA_	*a*ttribute type
CKC_	*c*ertificate type
CKD_	key *d*erivation function indicator in elliptic curve protocols
CKF_	bit *f*lag
CKG_	mask *g*eneration function indicator
CKH_	*h*ardware feature type
CKK_	*k*ey type
CKM_	*m*echanism type
CKN_	*n*otifications that cryptoki provides to an application
CKO_	*o*bject class
CKP_	*p*seudo-random function
CKS_	*s*ession state
CKR_	*r*eturn value
CKU_	*u*ser type
CKZ_	salt or encoding parameter source

Storage objects may also be separated following their lifespan. *Session objects* are destroyed as soon as the session in which they were created is closed; their visibility is limited to the application in which they were created. Contrary to session objects, *token objects* outlive the sessions in which they were created. When public, depending on the applied policy, token objects may be visible to all applications. When token objects are private, token secrete keys for instance, their visibility is reduced to authenticated applications.

The specification defines two types of users: the *Security Officer (SO)* and the *normal users*. The security officer's role is to administrate a token and its normal

users (termed *users* for short). The functions crafted for a SO include creation of a new user authentication mean (the *C_InitPIN* function), token reinitialization, i.e. the destruction of all objects that can be destroyed in a token (*C_InitToken*), etc. A SO cannot use a private object, only normal users can access the private objects.

PKCS #11 sessions. The communications between a user and a token are performed through a *session*. An application may have many sessions with a token, and a token may have different sessions with multiple applications. All the sessions of an application have the same state. Namely, if one session of an application succeeds in authentication, all the other sessions of that application becomes authenticated. And, if an application which already has a read–only session, say s_1, opens a read–write session s_2, the session s_1 becomes also a read–write one. Security Officer sessions are always read–write; a SO session can be either authenticated (R/W SO functions) or unauthenticated (R/W public session). User sessions can be either read–write or read–only. When an application opens an authenticated read–only session (R/O user functions), it has access to both public and private objects, but it cannot modify the private objects, or generate new ones. In an authenticated user read–write session, all token objects can be modified; object creation and destruction become also allowed. Notice also that, depending on implementation policy, a token may require a successful authentication prior to allowing access to any object — public or private.

5.3.2 Operations in the Standard Specification

More than sixty functions are provided in the standard specification; in addition implementor defined callbacks functions can be provided to Cryptoki for notification about some events. The callback functions are used, for instance, to inform about the already performed percentage of a potentially time–consuming function call. All the functions follow the "all–or–nothing" rule; information about the execution of a function and causes of failure (if any) are provided as an integer (which has to be interpreted). For conciseness, we group the PKCS #11 functions into three "families": container management, object management, and cryptographic use of the objects. A finer granularity description is given in [RSA04, pp. 89–187], however the general purpose functions, token management functions, and session management functions are all about containers (slots and tokens).

Container management. Container management functions include the *C_Initialize* function, which initializes a PKCS #11 library, the *C_GetSlotList* function to get the list of the available slots, *C_OpenSession* to open a session, etc. Container management functions can also provide information about a token (*C_GetTokenInfo*), a slot (*C_GetSlotInfo*), mechanisms implemented on a token or on a particular mechanism (*C_GetMechanismList, C_GetMechanismInfo*), or (un)authenticate a user (*C_Login, C_Logout*). Except the (un)authentication functions, the container management functions are not sensitive; the information they provide can be publicly available.

Object management. Object management functions are used to manipulate objects and their attributes. At a lower level, a PKCS #11 object is a collection of attributes; each attribute is a triple *(Type, Value, ValueLength)*, where *Type* is the attribute type (e.g. CKA_PRIVATE_EXPONENT for a RSA private exponent), *Value* is a byte array representing an arbitrary string of bytes, an integer, or an unpadded string with non–null termination, *ValueLength* is the length in bytes of *Value*. Object (including key) creation can be performed using the *C_CreateObject* function, with parameters a session handle, an object template, and a pointer to receive the created object's handle. When creating an object in this way, the provided template has to fill–in all the attribute values of the object, in addition to being consistent. The *C_CreateObject* function can be used, for instance, to import public keys or certificates into a token.

Key generation is performed using the *C_GenerateKey* or *C_GenerateKeyPair* functions with parameters a session handle, a key generation mechanism, the template of the key(s) to be generated, and pointer(s) to receive the handle(s) of the generated key(s); these functions generate respectively a symmetric key or a key pair. A symmetric key can also be derived from an existing one, using the *C_DeriveKey* function, with parameters the handle of the key to be derived and the key derivation mechanism.

Cryptographic use of objects. Storage objects are used with cryptographic functions. Cryptographic operations can be performed in different ways. An encryption operation is initialized using the *C_EncryptInit* function with parameters the encryption mechanism (e.g. CKM_AES_CBC), and the handle of the encrypting key. Once the operation initialized, the proper encryption can be performed using either the *C_Encrypt* function for a single–part data encryption or the *C_EncryptUpdate* and *C_EncryptFinal* functions for a multi–part data encryption. contrary to the *C_EncryptFinal* or *C_Encrypt* functions, the *C_EncryptUpdate* function can be called many times after a *C_EncryptInit* call; decryption, signature, signature verification, or digest related functions are defined in a similar way.

A user can import a *wrapped* (encrypted) key into a token using the *C_UnwrapKey* function with parameters the unwrapping key's handle, the unwrapping mechanism, a key template, and an address location to receive the unwrapped key's handle. Conversely, an *extractable* key can be exported off a token using the *C_WrapKey* function. In addition, the *C_GetAttributeValue* and *C_SetAttributeValue* functions can be used to read or modify the attributes of an object, if allowed; object destructions are performed using the *C_DestroyObject* function.

5.4 (In)Security in the PKCS #11 Standard

When "correctly" implemented, the specification constitutes a real basis for a secure cryptographic token interface. Unfortunately guaranteeing that an implementation is a secure one, with respect to a given threat model is not a trivial task. The security of a cryptographic device presents three aspects [WEI08]: (1) *logical security* which

is about the logical mechanisms to prevent unauthorized access to sensitive information; (2) *physical security* which is about the physical barriers to prevent unauthorized physical access to the computing system; and (3) *environmental security* which is the eventual protection mechanisms in the deployment environment of the device (access policy, cameras, etc.). However, physical and environmental security cannot be concretely discussed in detail apart from an implementation in a given deployment environment. In this section, we focus on the logical security in the PKCS #11 specification.

In the specification, accessing private objects is possible only after a successful authentication as a user; an attacker which gains physical access to a device and is unable to succeed in authentication cannot use the private objects or gather information about them. In addition a key can be marked as *sensitive* (the value of its CKA_SENSITIVE attribute is set to *true*) or *unextractable* (the value of the its CKA_EXTRACTABLE attribute is set to *false*). A sensitive key cannot be revealed in clear–text off its token, and an unextractable object cannot be extracted from its token, even in an encrypted form. In addition, the value of a CKA_SENSITIVE attribute can be modified only from *false* to *true*; similarly the value of an CKA_EXTRACTABLE attribute can change from *true* to *false*, but not in another way [RSA04, pp. 65, 130].

The standard specification argues that conforming with the restrictions on sensitive and unextractable keys in an implementation provides an adequate logical security for objects management. As, while operating system security concerns, and potential threats that may be caused by a malicious software or device driver, may allow an attacker to access a session, or compromise a PIN, if authentication is performed using a PIN, such attacks cannot compromise sensitive or unextractable keys, since *"a key that is sensitive will always remain sensitive"*; and *"a key that is unextractable cannot be modified to be extractable"* [RSA04, sect. 7, p. 31].

Unfortunately, in the standard specification, the discussion about the sensitive and unextractable objects in the presence of an attacker which accesses an authenticated session is incomplete. Since the important aspect for keys marked as unextractable or sensitive is not only to keep them marked unextractable or sensitive, but to guarantee that: (1) an unextractable key cannot be exported off its token, and (2) a sensitive key cannot be exported in clear–text off its token. Indeed, when an attacker accesses an authenticated session, the specification fails in circumventing many practical attacks [CLU03, DEL08]. In the following subsection, we recall some of these attacks.

5.4.1 Logical Security Weaknesses

The standard does no allow security officers to access private objects. This restriction is rather surprising; in addition to being not motivated in the specification, it is difficult to see how it can be achieved in practice; as a SO can create a valid authentication mean for itself (as a user), login as a user, and access private objects.

Sensitive Key Export. Symmetric keys can be used for key (un)wrapping. Recall that the allowed functional uses of a key are defined through the values of its attributes;

for instance, a key cannot be used for data encryption if the value of its CKA_ENCRYPT attribute is *false*. The specification differentiates data encryption from key wrapping for encrypting keys. Unfortunately, this distinction is not strict in cryptographic mechanisms; some mechanisms (CKM_AES_ECB, for instance) can be used for both key encryption and data decryption. In addition, the specification does not forbid any combination of the attributes values that indicate possible uses of a key. As a consequence, an attacker which accesses an authenticated read–write session (*R/W user functions*), in a PKCS #11 device implemented without further restrictions, can export in clear–text sensitive keys, using the following sequence of queries.

(a) Generate a symmetric key with the values of its CKA_WRAP and CKA_DECRYPT attributes set both to *true* (i.e. with ability to wrap keys and decrypt data).
(b) Wrap the target sensitive key, using the newly generated key as wrapping key, and using a mechanism which can be used also for data encryption, CKM_AES_ECB for instance (see [RSA04, pp. 188–192] for a summary of mechanisms uses).
(c) Decrypt the wrapped key (using the *C_Decrypt* function) to get the target sensitive in clear–text off the token.

As the specification allows keys to be used for both data encryption and key wrapping, the impossibility to extract sensitive keys in clear–text off their token cannot be guaranteed, unless further restrictions are applied to the implementing module. The separation of key roles should be clearly stated in the standard specification, together with the potential security consequences in a non–conform implementation.

Wrapping with a weaker key. If an attacker accesses an authenticated read only session (*R/O user functions*), with some computational effort, it may export in clear–text a target sensitive key, using the following heuristic sequence of queries.

(a) Search for a weak key with ability to wrap a key (if such a key is not found the attempts fails.)
(b) Wrap the target key using the weak key;
(c) If the target sensitive key can be used for encryption, then encrypt some data with it.
(d) Else, if it can be used for key wrapping, wrap the weak key, using the target key as wrapping key; else, the attempt fails.
(e) Perform a brute force attack on the weak key, and unwrap the target key to get it in clear–text.

If the target key can be used for data encryption, success in the brute force search can be tested using the encrypted data. Else, if the target key can be used for key wrapping, the brute force attack can be performed as follows. We denote the sensitive and weak keys by K_0 and K_1 respectively. The notation $\mathrm{Wrap}_T(\cdot,\cdot)$ denotes a wrapping operation performed in the token, the first parameter is the wrapping key; then $\omega_0 = \mathrm{Wrap}_T(K_1, K_0)$ and $\omega_1 = \mathrm{Wrap}_T(K_0, K_1)$ denote respectively the wrapped sensitive and weak keys. The notation $\mathrm{Unwrap}_{out}(\cdot,\cdot)$ denotes an unwrapping operation performed off the token.

For $K'_1 \in \{0,1\}^{|K_1|}$ **do**
$K'_0 = \text{Unwrap}_{out}(K'_1, \omega_0)$.
If $K'_1 = \text{Unwrap}_{out}(K'_0, \omega_1)$ **return** K'_0.
end for

The attacker simply tries all the possible values of the weak key K_1; when a key K'_1 such that $K'_1 = \text{Unwrap}_{out}(\text{Unwrap}_{out}(K'_1, \omega_0),\ \omega_1)$ is found (such a key is very likely K_1), $K'_0 = \text{Unwrap}_{out}(K'_1, \omega_0)$ is returned as K_0. Notice that the existence of weak keys in cryptographic devices is common; this is usually motivated by a need of compatibility with already existing infrastructures. The DES scheme [FIP99], for instance, even if considered now as a weak scheme due to the shortness of its keys, is implemented in many recent HSMs, including the Ultimaco SafeGuard CryptoServer [ULT09], which meets the FIPS 140–2 security level 3.

Key Space Reduction. Some of the key derivation mechanisms provided by the specification, make attacks possible against unextractable keys. Recall that unextractable keys cannot be extracted off its token, even if wrapped; and the value of an CKA_EXTRACTABLE attribute can change only from *true* to *false*. Unfortunately some key derivation mechanisms can be used to foil restrictions on unextractable keys.

The CKM_EXTRACT_KEY_FROM_KEY mechanism, for instance, permits to create a new key using parts of a base key. An attacker which gains access to an authenticated read–write session, can derive keys using separate parts of a target unextractable key, and perform separate exhaustive searches on the target key parts to recover the target key. For an unextractable 128–bit AES key the attacker can derive two DES keys, use the derived keys to encrypt some data, and perform separate exhaustive searches to find the derived DES keys; the remaining part of the target AES key can be found with a limited computational effort.

Key Wrapping and Fault Attacks. The *C_WrapKey* function can be used for fault attacks, in particular when the target key is a sensitive RSA private key, which can be used for signature; some of the RSA private key attributes are recalled in Table 5.2. Among the attributes in Table 5.2, only the CKA_MODULUS and CKA_PRIVATE_EXPONENT attributes are necessary for consistency; however, the other non–boolean attributes are commonly used for faster signature generation [BON92]. To know whether or not an RSA key has some sensitive attributes, one can call the *C_GetAttributeValue* function, with the object's handle and the template of the sensitive attributes one is asking about as parameter. Of course, the call will fail, as the attributes are sensitive. However, if the object has the attributes, the returned value is CKR_ATTRIBUTE_SENSITIVE, and CKR_ATTRIBUTE_TYPE_INVALID otherwise.

An attacker which accesses an authenticated read–write session, can wrap a RSA key with CKA_PRIME_1 and CKA_PRIME_2 attributes, using a mechanism which provide neither integrity checking nor error expansion (the CKM_AES_ECB mechanism for instance); it can then modify the wrapped key, to make one of the factors of the modulus n, say CKA_PRIME_1 or p, altered when unwrapped. It then follows a classical fault attack on RSA [BON92]. As the altered (wrapped) key can then

be unwrapped, and a signature generated using the faulty unwrapped key on some message, say M. The attacker may then recover a non–trivial factor of n using the signature and the message M, and then the private exponent of the target private key. As if S is a RSA signature on a M, generated using the altered unwrapped key, then

$$M' - (S)^e = M' - (S_q)^e = 0 \mod q,$$

where M' denotes the padded message before exponentiations, S_q is $S \bmod q$. Recall that M' can be computed for many signature mechanisms (the CKM_SHA384_RSA_PKCS mechanism, for instance). Better, for the CKM_RSA_X_509 mechanism, M' equals M. Now as S was generated using an erroneous factor p', it is likely that $M' - (S)^e = M' - (S_p)^e \neq 0 \bmod p$, and $\gcd(n, M' - (S)^e)$ yields a non–trivial factor of the modulus n, this allows in turn to recover the target private key.

Table 5.2: Some Attributes of a RSA Private Keys

Attribute	**Meaning**
CKA_MODULUS	an RSA modulus, usually termed n
CKA_PUBLIC_EXPONENT	public exponent e
CKA_PRIVATE_EXPONENT	private exponent d
CKA_PRIME_1	a prime factor p
CKA_PRIME_2	a prime factor q ($n = pq$)
CKA_EXPONENT_1	$d \bmod p - 1$
CKA_EXPONENT_2	$d \bmod q - 1$
CKA_COEFFICIENT	$q^{-1} \bmod p$
CKA_DECRYPT	boolean
CKA_SIGN	boolean

Other variants of the attacks we present are possible [CLU03, DEL08, CAC09], broadly the attacks highlight idiosyncrasies, from a security point of view, between functional and security goals of the PKCS #11 specification, the lack a tamper detection mechanism on wrapped keys, and the existence of mechanisms allowing the generation of related keys, with different uses.

5.4.2 Implementation solutions

The key wrapping function (*C_WrapKey*) is intended to the encryption and export of keys for backup storage, import in another token, or re–import in the source token at a later time. Unfortunately, as discussed above, the non–separation of key roles is problematic; key–wrapping keys can also be data encrypting keys, and yield sensitive key export attacks. Separating key roles partly circumvents such attacks, and more generally enhances implementation security.

Also, as most of the key wrapping mechanisms are also data encryption mechanisms, and then provide no tamper detection mechanism for wrapped keys, key wrapping based fault attacks are possible. Such attacks can be circumvented when keys are tagged prior to encryption (a digest of the wrapped key can be appended to the key

before it is encrypted). The weaker key wrapping attacks can also be circumvented, simply by forbidding the protection of a key with a weaker key.

The key derivation mechanisms can be problematic also, as the specification provides no restriction on the size or uses of a derived key. In addition some derivation mechanisms, CKM_EXTRACT_KEY_FROM_KEY or CKM_XOR_BASE_AND_DATA for instance, provide related keys such that a leakage on a derived key partially reveals the parent key. This security issue can be addressed by forbidding such key derivation mechanisms.

Also, separating key roles, i.e. guaranteeing that a key cannot have different contradictory roles (in regard to security) at the same time is not sufficient; it should also be guaranteed that key roles do not change during their lifespan. If a key role can change from wrapping key to data encrypting key, for instance, already wrapped keys can be decrypted as data and then exposed in clear–text off their token. And, if a data encrypting key can be modified to be a key wrapping key, previously encrypted data can potentially be unwrapped to get (token) sensitive keys with known value. In addition, a derived key should inherit the functional attributes (CKA_WRAP, CKA_DECRYPT, etc.) of its parent key. Since if not, an attacker which accesses an authenticated read–write session can derive two keys with the same key value, but with different roles, key wrapping and data encryption, for instance; and using the derived keys, export sensitive keys in clear–text.

The standard security weaknesses are well known [CLU03, CAC09, DEL08][2]. Unfortunately, the weaknesses cannot be circumvented without fundamental changes in the specification. As one can expect such changes come with the dual concerns of interoperability and backwards compatibility.

5.5 Overview of two PKCS #11 Implementations

In this section, we discuss few of the technicalities in our PKCS #11 implementations. Considering the commercial nature of the eKeynox^{TM} and RCP products the discussion is voluntarily limited.

5.5.1 Implementation for eKeynox^{TM}

Dedicated security devices attain a security level which cannot generally be achieved using general purpose computers. The eKeynox project deals with the design and management of such devices. In the eKeynox solution, cryptographic operations are performed using a PKCS #11 module; the components of the solution are briefly presented hereunder.

Overview of the eKeynox Components. The main physical components of the SMK are a microprocessor (ARM926 EJ–S [ARM]) which embeds a Linux (2.6.15/ ARM), a fingerprint sensor module (a TCS3C–TCD42 from UPEK [UPEK]), a flash

[2]Notice the long duration between the first version of the specification which was published in 1995, and the work of Clulow [CLU03], which is to our knowledge the first publication on PKCS #11 vulnerabilities.

memory and a chip (a DS3605 from Maxim [MAX]) for true random number generation. The random number generation chip and the fingerprint sensor are respectively connected to the microprocessor through its I2C [PHI95] and USB [AXE06] interfaces. The logical components of the system are as in Figure 5.4; our discussion focuses on the PKCS #11 interface bloc.

Figure 5.2: The "Smart Mobile Key"

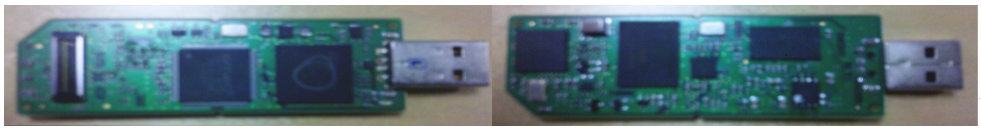


Figure 5.3: Overview of the SMK physical components

R. Serv.	:	Remote Servers,
Port. Apps.	:	Portable Applications,
Main App.	:	Main Application,
P11 Int.	:	PKCS #11 Interface,
CCL	:	Client Communication Library,
ESP	:	Embedded Services Provider,
FSM	:	Fingerprint Sensor Module.

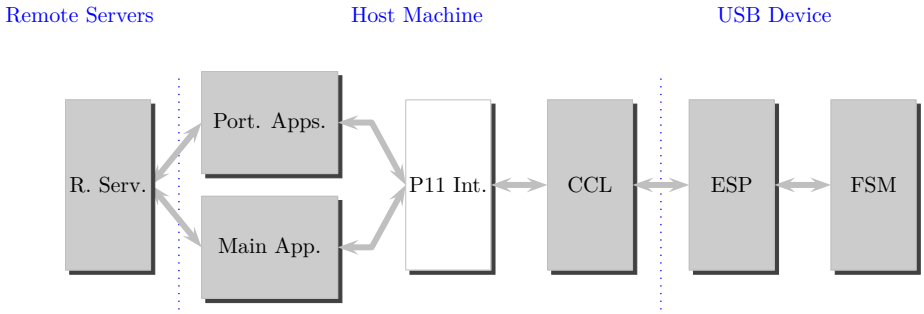


Figure 5.4: Main Modules in eKeynox

Overview of the PKCS #11 Module Internals. The type definitions provided in the specification are not all adequate for a practical implementation as they are. The sessions, for instance, are viewed off a token through handles (integer identifiers); internally sessions have to be represented with much more data to track session states, ongoing operations, generated objects, and so on. In addition internal structures

(objects, sessions, tokens, etc.) have to be structured in a manner which allows an efficient search. The structures from the specification are then internally added, with attributes (in the sense of the C language) to allow an easier tracking and use. Sessions, for instance, are internally represented as structures with attributes referencing the savings of session state and ongoing cryptographic operation states. Recall that an encryption can be performed using a sequence *C_EncryptInit*, *C_EncryptUpdate*, $\cdots$, *C_EncryptUpdate*, *C_EncryptFinal*, wherein the *C_EncryptUpdate* functions is called many times; it is then necessary to track ongoing operations. In addition, some internal structures are defined for convenience in guaranteeing the consistency of the internal structures accessed by a process. An overview of the internal structures interconnections is given in Figure 5.5; this color is used to indicate the internal structures which can be externally viewed through an identifier, and this one is used for structures with no external identifier, or internal references between structures.

```
typedef struct internal_session {
        CK_SLOT_ID slotId,
        CK_SESSION_HANDLE handle,
        CK_SESSION_INFO_PTR pSessionInfo,
        INTERN_OBJ_TABLE_PTR pObjectTable,
        CK_VOID_PTR pDecryptState,
        CK_VOID_PTR pSignState,
          ...
} INTERNAL_SESSION;
```

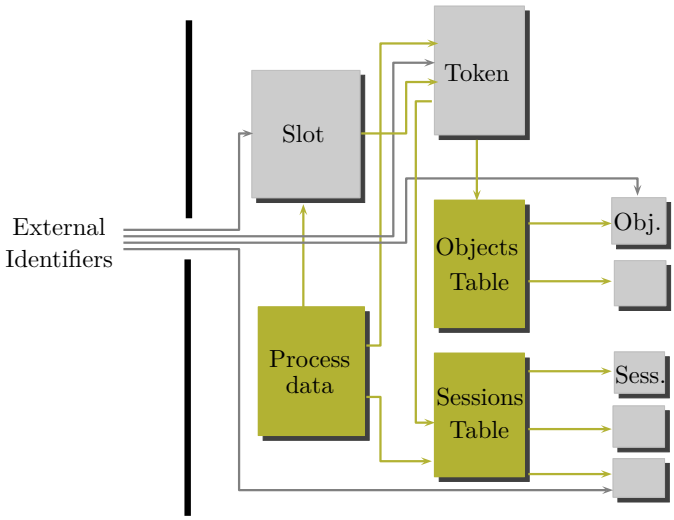


Figure 5.5: Interconnections between the Internal Structures

The *process data structure* is only for internal use in the module, it contains a process identifier (assigned to the application by host the operating system), and the identifiers of the sessions created by the process. The process data structure may also contain references to the functions to use for mutex (*mut*ual *ex*clusion) objects creation, destruction, or (un)locking. Recall that at the initialization of a PKCS #11

module, an application can indicate how the module should deal with the application's multi–threaded accesses to the module, and provide the addresses of the functions the module should use.

The *slot structure* contains generic slot information (slot description, version number, whether or not the token is present, etc.), and a reference to a token structure. The *token structure* contains information about the token (serial number, whether or not the token has its own random number generator, the available memory for private objects etc.), it also contains pointers to function implementations. Some functions are implemented in both the part of the PKCS #11 module which runns on the host machine, and in the embedded Linux. At function call, depending on the nature of the object to be used (session or token, public or private), the operation is performed either in the device or on the host machine, this design choice does not only lightweight the burden in the USB bus, it also makes possible the achievement of the security requirement that private token objects should never be used on the host machine. The *session structures* contain references to states of ongoing operations, and a reference to an objects table. The objects referenced in an object table are either the internal representations of the sessions objects, or internal handles which permit to use (embedded) token objects.

Authentication and Sensitive Data Protection. The authentication is twofold, in addition to scanning its fingerprint, the user provides a PIN code through the main application. Using the fingerprint and the PIN, an AES key is derived, and using the key, the token objects are decrypted, and the derived key destroyed. If the authentication fails, a counter is decreased, and when the counter reaches zero, the token is blocked; in this case the user has to contact the "help–desk operator" to unblock its token.

Figure 5.6: Authentication Screenshot

At token personalization phase, all the token objects needed to access the different servers or to use the device, together with a (256–bit) random number are gener-

ated, wrapped and saved on a dedicated server. The random number is latter used by the helpdesk operator for to update the user's certificates. In addition, after the personalization phase, no token key generation or derivation is allowed, except those periodically performed by a helpdesk operator for an update of the certificate. There is no special physical protection for the token objects; however, except at the personalization phase, the token keys cannot be exported off a token (all the functions calls dealing with token objects export are rejected). The token objects functional attributes are separated and cannot be changed. The schemes considered as weak (DES, RC2, etc.,) are not implemented, as the eKeynox solution needs no already existing infrastructure, the compatibility issues are minored.

When a device owner is not authenticated, the token objects are encrypted with a key which cannot be derived unless its fingerprint in PIN code are provided. And, except at personalization phase, or during an update of a certificate, the token objects are "read–only." Hence one can have reasonable confidence in the physical security provided by the implementation.

To enhance the logical security, the applications embedded in the environment are modified to forbid temporary files or debug information logs on the host machine; the necessary temporary files are created in the device secure storage. In addition, before a portable application is launched by the main application, a (256–bit) random number is generated in the embedded Linux, and injected in the application at launch. When the applications calls a function which uses the token objects, it provides the random number as additional (implicit) parameter; at the PKCS #11 module the calling process identifier is also appended to the parameters. The random number becomes obsolete when the application closes. With these mechanisms one can have a reasonable confidence on the logical and physical security of the PKCS #11 module.

Performance Issues. When a cryptographic operation deals only with session objects, it is performed in the calling process address space, and with most of the host machines there is no performance issue for session objects. Indeed for session objects, the implementation uses the OpenSSL low–level cryptographic API [APV07]; a 2048–bit RSA signature is performed within a few dozens of milliseconds (on Windows XP Intel Core2 processor, 2.13GHz CPU). Unfortunately, for token objects, the efficiency decreases very significantly; a 2048–bit RSA key pair generation takes 29 seconds, and a signature generation 1.6 seconds. Token key pair generation is however performed only at personalization phase, or for certificate update.

This eKeynox solution, is today used on tokens compatible with Windows 2000, Windows XP, and Vista, in many companies.

5.5.2 Implementation for RCP

The RCP project aims to provide a generic and modular cryptographic design together with VHDL and C source codes for a rapid prototyping of cryptographic devices, depending on costumers needs. Broadly, besides genericity and modularity, the feature requests in he RCP project include efficiency, price scalability, and naturally security

scalability. The outcome of the project should be a proof of concept which is resilient to both logical and physical attacks. In addition objects backup, restore, and firmware update mechanisms have to be provided; and a protected authentication path, whereby a user can directly log into a token, without using the host machine, have to be designed.

In this section, we discuss some of the of design choices and the implementation aspects. Notice that we do not discuss the reconfiguration aspects; a discussion on this point can be found in [BAD09], our work on RCP was performed together with this author.

Design Overview. The main components in the RCP proof of concept (POC) are a secure Processor (the "Universal Secure Integrated Platform" (USIP) from Innova card — which is now part of Maxim [MAX]), a FPGA (a Virtex 5) a smart card reader, a flash memory, and a PCIe connector (see Figures 5.7 and 5.8).

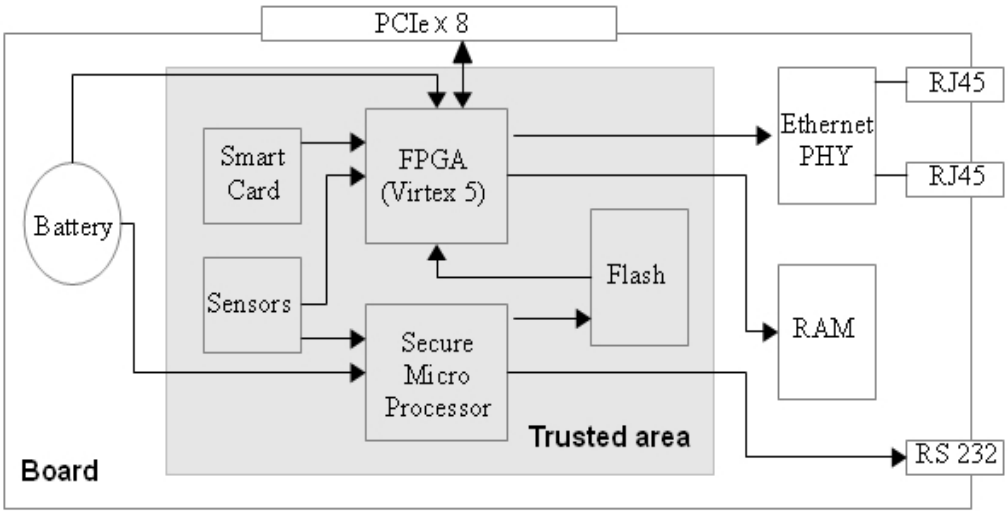


Figure 5.7: RCP Board Block Diagram

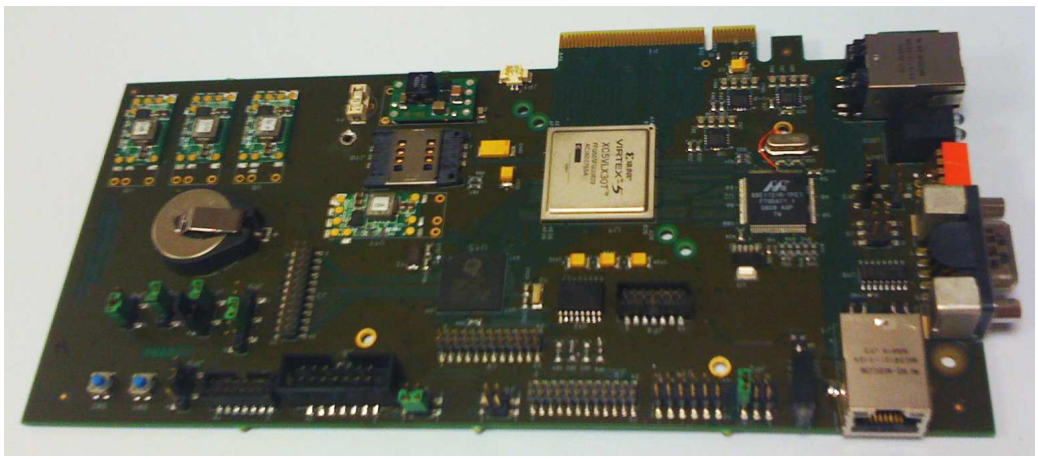


Figure 5.8: RCP Board

Figure 5.9: The SCM Microsystems SPR532 PINpad

The FPGA (field programmable gate array) is used for performance scalability; it serves for the core cryptographic engine implementations, so lower or higher performance FPGAs can be used depending on the needs. (It serves also as a lever for the board price — the USIP price, 20$, is relatively low). FPGAs have no non–valatile memory and cryptographic engines only are not sufficient, if not coupled with a secure key management. The USIP provides a true RNG, integrated tamper sensors (frequency, voltage, temperature, and die shield), external sensor inputs, a keyboard controller, four memories (128K bytes of SRAM, 256K bytes of flash memory, 256 bytes of user one–time programmable memory, and 128K bytes of ROM), and an "on the fly AES encryption" of external memories. This represents a non negligible security add–on; the USIP is used for master keys generation and storage.

For communication flexibility the FPGA is connected to a PCIe bus with 8 lanes with a maximal throughput of 16 gigabit per second; the FPGA is also connected to two RJ–45 interfaces that can support gigabit Ethernet. The communication interfaces were voluntarily chosen to allow the use of different paths for plaintext and ciphertext (red/black architecture). The smart–card reader is integrated for direct authentication on the device; in addition the RS–232 interface, supported by the open source smart card projects (OpenSC[3] for instance) can be used to connect the PINpad (a SPR532 from SCM Microsystems [SCM]). The USIP provides tamper respondent mechanisms to fulfill the security requirements of the FIPS 140–2 level 3. However, many FPGAs cannot achieve the level 3 requirements, the reason is simply that they provide no tamper respondent mechanism, invasive attacks are then possible. To make the RCP possibly achieve the level 3 requirements, the trusted area is enclosed with a tamper respondent sensor surface connected to the USIP external sensors interface.

With these design choices, the USIP acts as the "master" of the system, and the FPGA as an accelerator which responds to the master's calls.

On the RCP board, are mapped the abstraction layers summarized in Figure 5.10. We defined a complementary API, which deals with the function requests which are not defined in the PKCS #11 interface. It allows a backup of the whole objects contained the system, and an update embedded C or VHDL codes. It allows also audit trial exports, and system uninstall (the latter operation destroys all objects and makes the system identical to an ex–works one).

[3]See http://www.opensc-project.org/.

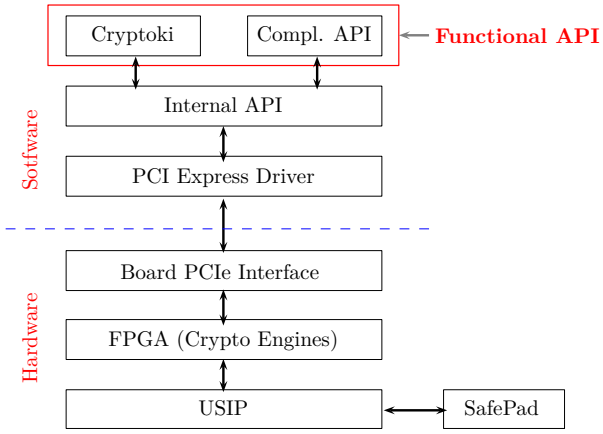


Figure 5.10: RCP abstraction layers

Key Management. As usual the ex–works boards will be empty. The empty boards are first pre–initialized by Netheos, which loads the FPGA bitstream and related materials for bitstream confidentiality and secure remote updates. The secure flash loader key of the USIP is then initialized, and the USIP code is encrypted with the flash loader key, and then loaded in the embedded flash memory (which is different from the USIP internal flash memory).

The proper initialization is performed in the deployment environment, the user runs a simple application provided by Netheos, and insert smart–cards when required for a backup of the master keys, and users authentication smart-cards. Internally the initialization is also rather simple, the USIP generates the master keys, read the time and date from the user, and stores it in its real time counter (RTC) for stamps generation; the USIP then generates an empty record of the functional keys. Latter, when a user queries a key generation, the key is generated and encrypted in the USIP, and copied in the backup flash partition; finally the USIP updates the record of valid keys to include the newly generated key. When a user issue an operation which involves a key, the USIP decrypts the key, checks it's validity an transmits it to the FPGA which performs the operation; the private keys are never stored in cleartext out of the trusted area. The communications between the FPGA and the USIP are secured with a symmetric key which is injected in the FPGA bitstream and USIP running code at pre–initialization.

For an external backup of the functional keys, only allowed to SOs, a key is generated and distributed on some smart cards, the backup flash partition is decrypted, and checked for integrity, and then MACed together with the digest of the keys to be backuped and encrypted with the newly generated key. Finally the backup "blob" is copied in the indicated location.

Next Steps in the Development. The hardware specifications was defined and a first version of the board was released in the end of 2008. At this writing, some parts of the complementary library and VHDL code are still under development; and the tamper responder enclosure is not integrated yet.

The PKCS #11 API is added with the restrictions discussed in subsection 5.4.2 to resist logical attacks. Without surface enclosure, while simple side–channel attacks, simple power analysis or timing attack, are considered the core engines development, the board may be subject to differential power analysis or fault injection. Including the secure enclosure is mandatory. The USIP is configured to respond by erasing its battery powered key at sensors alarm; in this case, the master key is lost and the board keys are lost also. The already developed parts will be completed and used soon.

Chapter 6

Conclusion

Our work focused on the topic of security models for authenticated key agreement and the design of authenticated key agreement protocols. We analyzed the main security models for the analysis of Diffie–Hellman protocols in the public key setting, the CK and eCK models, and provided secure and efficient designs of key agreement protocol.

Explored topics. In this dissertation, we recalled the Bellare–Rogaway model, and presented the CK and eCK security models. We showed how the CK matching sessions definition makes some protocols, shown secure fail in authentication. We illustrated this, using the protocol $\mathcal{P}$, which we showed to fail in authentication when leakages on ephemeral Diffie–Hellman exponents are considered. We described the eCK model, and showed that its ephemeral key definition, combined with the use of the NAXOS transformation, yields protocols which may be formally secure, but practically insecure, as vulnerable to ephemeral Diffie–Hellman exponent leakages. Taking the aforementioned analysis into account, we proposed a new security model the *strengthened eCK model* (seCK), which encompasses the eCK model. Moreover, contrary to the eCK model which considers that an attacker may learn the ephemeral secret key at a party, independently of the way in which the protocol is implemented, the hypothesis in the seCK model is that an attacker may learn any information at a party, except those stored in a tamper resistant device. The seCK model is not only about the formal treatment of key agreement protocols, it is also about securely implementing authenticated key exchange. When a protocol is shown secure in the seCK model, it is also specified how it can be *securely and efficiently implemented.* Providing security arguments in the seCK model, includes stating which operations or data have to be handled in an area with the same protection mechanisms as the area in which is stored the session owner's static key, and which information can be computed or stored in an untrusted area.

We identified new points, termed i–points, which can be used for impersonation and man–in–the–middle attacks against the (H, C)MQV(-C) protocols, and explored the search of these points. The method we propose for decomposed i–point search is expected to be twice faster than the Pollard's rho algorithm. We provided a complementary analysis of the (H)MQV protocols, which are possibly the most efficient two–party authenticated Diffie-Hellman protocols in the public key setting, and showed these protocols vulnerable, to session specific information leakages. We proposed two new building blocks for Diffie–Hellman protocols, the FXCR–1 and FXCR schemes, and using these building blocks we proposed FHMQV and SMQV protocols, which provably meet the seCK security definition, under the random oracle model and the Gap Diffie–Hellman Assumption.

As the FHMQV and SMQV protocols are resilient to intermediate results leakages, their implementations can be made particularly efficient in environments wherein exist a tamper resistant device, together with a host machine, which may be untrusted. The ephemeral keys can be computed in the device in idle–time, while the costly exponentation in session key generation performed on the host machine. When the underlying group is that of the rational points of an elliptic curve, the FHMQV and SMQV can be securely implemented using computationally limited smart–cards. This allows efficient and secure implementations of elliptic curve based protocols, using computationally limited and (potentially low price) tamper resistant devices.

We also worked on two PKCS #11 implementations sketched in chapter 5. Even if the communications on these works was limited, these implementations were a real opportunity to face technical problems that may raise a practical implementation of cryptographic schemes.

Open issues and future research directions. It is worth to explore further possible optimizations in the parallelizations of the decomposed i–point search. When using the Montgomery's simultaneous inversion technique together with the López–Dahab affine formulas, inversions can be significantly reduced; however, with regard to the remaining inversion and multiplications the parallelized decomposed i–point search does not seem applicable for cryptographic curves, unless added with further significant optimizations. Nevertheless, this approach is more easily parallelizable than the Pollard's rho algorithm. As, when the decomposed i–point search is parallelized, no shared storage is needed, and no communication is needed between the processors, except at initialization, or when a processor finds a decomposed i–points and informs the other processors.

There is no well established paradigm for the design of authenticated key agreement protocols. And as can be noticed, there is an underlying identification scheme, which may be implicit or explicit, in any authenticated key exchange. This raises the question of the properties an identification scheme should have to allow the design of a provably secure authenticated key agreement protocol. An answer to this question would reduce the design of an authenticated key agreement protocol to that of an identification scheme.

As future prospects, we will be interested in possible optimizations in decomposed i–point search. We will also give attention to the question of the conditions an identification scheme must satisfy to yield a provably secure protocol. Our work focused mainly on Diffie–Hellman protocols in the public key setting, we will be interested in the adaptation of ideas we developed in the public key setting to password based protocols.

Related Publications and Reports

(in chronological order)

- Sarr A. P.: Analysis of the ECMQV Protocol. Technical report, 2008.
- Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A Secure and Efficient Authenticated Diffie–Hellman Protocol. In Proc. of Public Key Infrastructures, Services and Applications — EuroPKI 2009, Lecture Notes in Computer Science, vol. 6391, pp. 83–98, Springer–Verlag, 2010.
- Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A Secure and Efficient Authenticated Diffie–Hellman Protocol (extended version). Cryptology ePrint Archive, Report 2009/408, 2009.
- Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: Enhanced Security and Efficiency for Authenticated Key Agreement. International Workshop on Foundations of Security and Privacy — FCS-PrivMod 2010.
- Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A New Security Model for Authenticated Key Agreement (extended version). Cryptology ePrint Archive, Report 2010/237, 2010.
- Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A New Security Model for Authenticated Key Agreement. In Proc. of the 7th International Conference on Security and Cryptography for Networks — SCN 2010, Lecture Notes in Computer Science, vol. 6280, pp. 219–234, Springer–Verlag, 2010.

Bibliography

[ABD05] Abdalla M.,Fouque P.A., Pointcheval D.: Password-based authenticated key exchange in the three-party setting. In Proc of PKC 2005, Lecture Notes in Computer Science, vol. 3386, pp. 65–84, Springer-Verlag, 2005.

[AND94] Anderson R.: Why Cryptosystems Fail. Communications of the ACM, vol. 37, pp. 215–227 ACM, 1994.

[ANS01a] ANSI X9.42.: Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography. ANSI, 2001.

[ANS01b] ANSI X9.63.: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography. ANSI, 2001.

[ANS05] ANSI X9.62.: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA). ANSI, 2005.

[APV07] Apvrille A.: Conception et architecture de la bibliothèque cryptographique d'OpenSSL. MISC number 32, pp.52–60, July 2007.

[ARN93] Arno S., Wheeler. F. S.: Signed Digit Representations of Minimal Hamming Weight. IEEE Transactions on Computers vol. 42(8), pp. 1007–1010, IEEE, 1993.

[AVA05] Avanzi R.: A note on the signed sliding window integer recoding and a left-to-right analogue. In Proc. of Selected Areas in Cryptography 2004, Lecture Notes in Computer Science, vol. 3357, pp. 130–143. Springer–Verlag, 2005.

[AXE06] Axelson J.: USB Mass Storage Designing and Programming Devices and Embedded Hosts. Lakeview Research LLC Madison, 2006.

[BAD09] Badrignans B.: Using FPGAs for Security–Sensitive Applications. PhD thesis, Université Montpellier 2, 2009.

[BAI08] Bai S., Brent R. P.: On the Efficiency of Pollard's Rho Method for Discrete Logarithms. InProc. Fourteenth Computing — The Australasian Theory Symposium CATS, vol. 37, pp. 125–131, Australian Computer Society, 2008.

[BAIL98] Bailey D. V., Paar C. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In Proc. of Crypto 98, Lecture Notes of Computer Science, vol. 1462, pp. 472–485, 1998.

[BAIL01] Bailey D. V., Paar C. Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography. Journal of Cryptology, vol. 14, pp. 153–176, 2001.

[BAS10] Basin D., Cremers C.: Modeling and Analyzing Security in the Presence of Compromising Adversaries. In Proc. of ESORICS 2010, Lecture Notes in Computer Science, vol. 6345, pp. 340–356, 2010.

[BEL98] Bellare M., Canetti R., Krawczyk H.: A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In Proc. of the thirtieth annual ACM symposium on Theory of computing, pp. 419–428, ACM, 1998.

[BEL04] Bellare M., Palacio A.: The Knowledge–of–Exponent Assumptions and 3–round Zero–Knowledge Protocols. In Proc. of Crypto 04, Lecture Notes in Computer Science, vol. 3152, 273–289, Springer–Verlag, 2004.

[BEL00] Bellare M., Pointcheval, D., Rogaway P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In Proc. of EUROCRYPT 2000, Lecture Notes in Computer Science, vol. 1807, pp. 139–155, Springer–Verlag, 2000.

[BEL93a] Bellare M., Rogaway P.: Entity Authentication and Key Distribution. In Proc. of Crypto 93, Lecture Notes in Computer Science, vol. 773, pp. 232–249, Springer–Verlag, 1993.

[BEL93b] Bellare M., Rogaway P.: Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In Proc. of the 1st ACM Conference on Computer and Communications Security, pp. 62–73, ACM, 1993.

[BEL95] Bellare M., Rogaway P.: Provably Secure Session Key Distribution — The Three Party Case. In Proc. of the twenty–seventh annual ACM symposium on Theory of computing, pp. 57–66, ACM, 1995.

[BEL97] Bellare M., Rogaway P.: Minimizing the Use of Random Oracles in Authenticated Encryption Schemes. In Proc. of the First International Conference on Information and Communication Security, Lecture Notes In Computer Science, vol. 1334, pp. 1–16, Springer Verlag, 1997.

[BEN09] Bender J., Fischlin M., Kügler D.: Security Analysis of the PACE Key-Agreement Protocol, In Proc. of ISC 2009, Lecture Notes in Computer Science, vol. 5735, pp. 33–48, Springer Verlag, 2009.

[BER08] Bernstein D., Lange T.: Faster Addition and Doubling on Elliptic Curves. In Proc. of Asiacrypt 2007, Lecture Notes in Computer Science, vol. 4833, pp. 29–50, Springer–Verlag, 2008.

[BIL03] Billet O., Joye M.: The Jacobi Model of an Elliptic Curve and Side-Channel Analysis. In Proc of Applied Algebra, Algebraic Algorithms and Error-Correcting Codes — AAECC 2003, Lecture Notes in Computer Science, vol. 2643, pp. 34–42, Springer–Verlag, 2003.

[BLA00] Blake I. F., Seroussi G., Smart N. P.: Elliptic Curves in Cryptography, London Mathematical Society Lecture Note Series, vol. 265, Cambridge University Press, 2000.

[BLA97a] Blake–Wilson S., Johnson D., Menezes A.: Key Agreement Protocols and their Security Analysis. In Proc. of the 6th IMA International Conference on Cryptography and Coding, Lecture Notes of Computer Science, vol. 1355, pp. 30–45, Springer–Verlag, 1997.

[BLA97b] Blake–Wilson S., Menezes A.: Security Proofs for Entity Authentication and Authenticated Key Transport Protocols Employing Asymmetric Techniques. Lecture Notes of Computer Science, vol. 1361, pp. 137–158, Springer–Verlag, 1997.

[BLA99] Blake–Wilson S., Menezes A.: Unknown Key–Share Attacks on the Station–to–Station (STS) Protocol. In Proc. of the Second International Workshop on Practice and Theory in Public Key Cryptography Lecture Notes of Computer Science, vol. 1560, pp. 154–170, Springer–Verlag, 1999.

[BLU84] Blum M., Micali S.: How to Generate Cryptographically Strong Sequences of Pseudorandom Bits. SIAM Journal on Computing, vol. 13, pp. 850–864, 1984.

[BON92] Boneh D.: Twenty Years of Attacks on the RSA Cryptosystem. Notices of the American Mathematical Society, vol. 46(2), pp. 203–212, 1999.

[BON01] Boneh D., Shparlinski I.: On the Unpredictability of Bits of the Elliptic Curve Diffie–Hellman Scheme. In Proc. of Crypto 01, Lecture Notes in Computer Science, vol. 2139, pp. 201–212, Springer–Verlag, 2001.

[BOS09] Bos J., Kaihara M., Montgomery P.: Pollard rho on the PlayStation 3. Handouts of SHARCS 2009, pp. 35–50, 2009.

[BOS10] Bos J., Kleinjung T., Lenstra A.: On the Use of the Negation Map in the Pollard Rho Method. In Proc. of the Ninth Algorithmic Number Theory Symposium, Lecture Notes in Computer Science, vol. 6197, pp. 67-83, 2010.

[BOY03] Boyd C., Mathuria A.: Protocols for Authentication and Key Establishment. Springer Verlag, 2003.

[BRE02] Bresson E., Chevassut O., Pointcheval D.: Dynamic Group Diffie–Hellman Key Exchange under Standard Assumptions. In Proc. of Eurocrypt 02, Lecture Notes in Computer Science, vol. 2332, pp. 321–336, Springer–Verlag, 2002.

[BRI02] Brier É. Joye M.: Weierstrass Elliptic Curves and Side-Channel Attacks. Public Key Cryptography – PKC 2002, Lecture Notes in Computer Science, vol. 2274, pp. 335–345, Springer–Verlag, 2002.

[BRO05] Brown D.: Generic groups, collision resistance, and ECDSA. Designs, Codes and Cryptography, vol. 35 (1), pp. 119–152, 2005.

[BSI10] BSI: Advanced Security Mechanism for Machine Readable Travel Documents — Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI), TR–03110, Version 2.03, 2010 (available at https://www.bsi.bund.de.)

[CAC09] Cachin C., Chandrany N.: A Secure Cryptographic Token Interface. In Proc. of the 22nd IEEE Computer Security Foundations Symposium, pp. 141–153, IEEE Computer Society (available from http://www.zurich.ibm.com/~cca/papers/mkms.pdf) 2009.

[CAN01] Canetti R., Krawczyk H.: Analysis of Key–Exchange Protocols and Their Use for Building Secure Channels. Lecture Notes of Computer Science, vol. 2045, pp. 453–474, Springer–Verlag, 2001.

[CAN01a] Canetti R., Krawczyk H.: Analysis of Key–Exchange Protocols and Their Use for Building Secure Channels. Cryptology ePrint Archive, 2001/040, 2001.

[CAN02] Canetti R., Krawczyk H.: Security Analysis of IKE's Signature–based Key–Exchange Protocol. In Proc of Crypto 02, Lecture Notes in Computer Science, vol. 2442, pp. 143–161. Springer–Verlag, 2002 (extended version available from http://eprint.iacr.org/2002/120).

[CERT09] Certicom Corp.: Certicom ECC Challenge, available at http://www.certicom.com/images/pdfs/challenge-2009.pdf, 2009.

[CHE03] Chen L., Kudla C.: Identity Based Authenticated Key Agreement Protocols from Pairings. In Proc. of in 16th IEEE Computer Security Foundations Workshop, pp. 219–233, IEEE, 2003 (Revised version at http://eprint.iacr.org/2002/184/).

[CHO05] Choo K. R., Boyd C., Hitchcock Y.: Examining Indistinguishability–Based Proof Models for Key Establishment Protocols. Lecture Notes in Computer Science, vol. 3788, pp. 624–643, Springer–Verlag, 2005.

[CLU03] Clulow J.: On the Security of PKCS #11. Lecture Notes in Computer Science, vol. 2779, pp. 411–425, Springer–Verlag, 2003.

[COH05a] Cohen H.: Analysis of the Flexible Window Powering Algorithm. Journal of Cryptology, vol. 18(1), pp. 63–76, 2005.

[COH05b] Cohen H, Frey G. (editors): Handbook of Elliptic and Hyperelliptic Curve Cryptography, CRC Press, 2005.

[COR01] Coron J.S., Naccache D., Kocher P.: Statistics and Secret Leakage. Lecture Notes in Computer Science, vol. 1962, pp. 157–173, Springer–Verlag, 2001.

[COR08] Coron J.S., Patarin J., Seurin Y.: The Random Oracle Model and the Ideal Cipher Model are Equivalent. In Proc. of Crypto 08, Lecture Notes in Computer Science, vol. 5157, pp. 1–20, Springer–Verlag, 2008.

[CRA04] Cramer R., Shoup V.: Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. SIAM Journal on Computing, vol. 33(1), pp. 167–226, 2004.

[CRE09a] Cremers C.: Session–state Reveal is stronger than Ephemeral Key Reveal: Attacking the NAXOS key exchange protocol. Lecture Notes in Computer Science, vol. 5536, pp. 20–33, Springer–Verlag, 2009.

[CRE09b] Cremers C.: Formally and Practically Relating the CK, CK-HMQV, and eCK Security Models for Authenticated Key Exchange. Cryptology ePrint Archive, 2009/253, 2009.

[DEL08] Delaune S., Kremer S., Steel G.: Formal Analysis of PKCS #11. In Proc. of the 21st IEEE Computer Security Foundations Symposium, pp. 331–344, IEEE Computer Society, 2008.

[DEN88] Den Boer B.: Diffie–Hillman is as Strong as Discrete Log for Certain Primes. In Proc of Crypto 88, Lecture Notes in Computer Science, vol. 403, pp. 530–539, Springer–Verlag, 1990.

[DEN02] Dent A.: Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model. In Proc. of Asiacrypt 02, Lecture Notes in Computer Science, vol. 2501, pp. 100–109, Springer–Verlag, 2002.

[DIF76] Diffie W., Hellman M. E.: New Directions in Cryptography. IEEE Transactions on Information Theory, vol. 22(6), pp. 644–654, IEEE 1976 (available from http://dret.net/biblio/reference/dif77).

[DIF92] Diffie W., van Oorschot P. C., Wiener M. J.: Authentication and Authenticated Key Exchanges. Designs, Codes and Cryptography, vol. 2(2), pp. 107–125, Springer Netherlands, 1992.

[DRIO53] Drioton É.: Les principes de la cryptographie égyptienne. Comptes–rendus des séances de l'année – Académie des inscriptions et belles–lettres, 97e année, N. 3, pp. 355-364, 1953 (available from http://www.persee.fr).

[DUQ07] Duquesne S.: Improving the arithmetic of elliptic curves in the Jacobi model. Information Processing Letters, vol. 104(3), pp. 101–105, Elsevier, 2007.

[EDW07] Edwards H.: A normal form for elliptic curves. Bulletin of the American Mathematical Society, vol 44(3), pp. 393–422, AMS, 2007.

[FIA86] Fiat A., Shamir A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. Lecture Notes in Computer Science, vol. 263, pp. 186–194, Springer–Verlag, 1986.

[FIP08] FIPS 180–3: Secure Hash Standard (SHS), FIPS 2008 (available from http://csrc.nist.gov/publications/PubsFIPS.html.)

[FIP00] FIPS 186–2: Digital Signature Standard (DSS). 2000 (available from http://csrc.nist.gov/publications/PubsFIPS.html.)

[FIP01] FIPS 140–2: Security Requirements for Cryptographic Modules. 2001 (available from (available from http://csrc.nist.gov/publications/PubsFIPS.html.)

[FIP99] FIPS 46–3: Data Encryption Standard (DES). 1999.

[FOU00] Fouquet M., Gaudry P., Harley R.: An extension of Satoh's algorithm and its implementation. Journal of the Ramanujan Mathematical Society, vol. 15(4), pp. 281–318, 2000.

[FRE94] Frey G., Rück H.: A remark concerning m–divisibility and the discrete logarithm in the divisor class group of curves. Mathematics of Computation, vol. 62, pp. 865–874, 1994.

[FRE98] Frey G., Gangl H.: How to disguise an elliptic curve. Talk at Waterloo workshop on the ECDLP, 1998 (available at http://www.cacr.math.uwaterloo.ca/conferences/1998/ecc98/slides.html.

[GAL00] Gallant R., Lambert R., Vanstone S.: Improving the Parallelized Pollard Lambda Search on Binary Anomalous Curves. Mathematics of Computation, vol. 69(232), pp. 1699–1705, American Mathematical Society, 2000.

[GAU02] Gaudry P., Hess F., and Smart N. P.: Constructive and Destructive Facets of Weil Descent on Elliptic Curves. Journal of Cryptology, vol. 15(1), pp. 19–46, 2002.

[GOL84] Goldwasser S., Kilian J.: Almost all primes can be quickly certified. In Proc. of the 18th Annual ACM Symposium on Theory of Computing, pp. 316–329, ACM, 1986.

[GOL84] Goldwasser S., Micali S.: Probabilistic Encryption. Journal of Computer and System Sciences, vol. 28, pp. 270–299 1984.

[GOL88] Goldwasser S., Micali S., Rivest R.: A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. SIAM Journal on Computing, vol. 17, pp. 281–308,1988.

[GOP07] Gopalakrishnan K., Thériault N., Yao. C. Z.: Solving Discrete Logarithms from Partial Knowledge of the Key. Lecture Notes in Computer Science, vol. 4859, pp. 224–237, Springer–Verlag, 2007.

[HAN03] Hankerson D., Menezes A., Vanstone S.: Guide to Elliptic Curve Cryptography. Springer–Verlag, 2003.

[IBM08] IBM Corp.: CCA Basic Services Reference and Guide for the IBM 4758 PCI and IBM 4764 PCI–X Cryptographic Coprocessors, 19th ed., IBM, 2008 (available from http://www-03.ibm.com/security/cryptocards/pcicc/library.shtml.)

[IEE00] IEEE 1363: Standard Specifications for Public Key Cryptography., IEEE, 2000.

[IEE09] IEEE 1363: Standard Specifications for Public Key Cryptography (Working Draft). IEEE, 2009.

[INP09] INPI: Le brevet, tout ce qu'il faut savoir avant de déposer un brevet. INPI, 2009 (available from http://www.inpi.fr.)

[ISO02] ISO/IEC IS 9798–3: Information Technology – Security techniques – Cryptography techniques based on elliptic curves – Part 3: Key Establishment., 2002.

[JEO04] Jeong I., Katz J., Lee. D.: One–round protocols for two–party authenticated key exchange. Lecture Notes in Computer Science, vol. 3089, pp. 220–232, Springer–Verlag, 2004.

[JEV58] Jevons W. S.: The Principles of Science: A Treatise on Logic and Scientific Method p. 141, Macmillan & Co., London, 1874; more recently reprinted by Dover Publications, New York, NY, 1958 (available at http://www.archive.org/stream/principlesofscie00jevorich#page/n166/mode/1up)

[JUS96] Just M., Vaudenay S.: Authenticated Multi–Party Key Agreement. Lecture Notes in Computer Science vol. 1163, pp. 36–49, Springer–Verlag, 1996.

[KAL01] Kaliski B.: An unknown key–share attack on the MQV key agreement protocol. ACM Transactions on Information and System Security (TISSEC), vol. 4(3), pp. 275–288, ACM, 2001.

[KIM09] Kim M., Fujioka A., Ustaoglu B.: Strongly Secure Authenticated Key Exchange without NAXOS' Approach. In Proc. of the fourth International Workshop on Security, IWSEC 09, Lecture Notes in Computer Science, vol. 5824, pp. 174–191, Springer–Verlag, 2009.

[KOC96] Kocher P.: Timing attacks on implementations of Diffe–Hellman, RSA, DSS, and other systems. Advances in Cryptology – Crypto 96, Lecture Notes in Computer Science, vol. 1109, pp. 104–113, Springer–Verlag, 1996.

[KNU81] Knuth D.: The Art of Computer Programming. vol. 2. Addison Wesley, Reading Massachusetts, 1981.

[KOB87] Koblitz N.: Elliptic Curve Cryptosystems. Mathematics of Computation, vol. 48(177), pp. 203–209, 1987.

[KOB07a] Koblitz N.: The Uneasy Relationship Between Mathematics and Cryptography. Notices of the AMS, vol. 54(8), pp. 973–979, AMS, 2007.

[KOB07b] Koblitz N., Menezes A.: Another Look at "Provable Security". Journal of Crypptology, vol. 20(1), pp. 3–37, 2007.

[KRA05] Krawczyk H.: HMQV: A Hight Performance Secure Diffie–Hellman Protocol (full version). Cryptology ePrint Archive, 2005/176, 2005.

[KRA05b] Krawczyk H.: HMQV: A High-Performance Secure Diffie-Hellman Protocol. In proc. of Crypto 05, Lecture Notes in Computer Science, vol. 3621, pp. 546–566, Springer–Verlag, 2005.

[KRA06] Krawczyk H.: Method and Structure for Challenge–Response Signatures and Hight–Performance Secure Diffie–Hellman Protocol., US Patent Application Publication 0179319 A1, 2006.

[KRA08] Krawczyk H.: Method and Structure for Challenge–Response Signatures and Hight–Performance Secure Diffie–Hellman Protocol., European Patent 1847062 B1, 2008.

[KUN06] Kunz-Jacques S., Pointcheval D.: About the Security of MTI/C0 and MQV. In proc. of the international conference Security and Cryptography for Networks, SCN 2006, Lecture Notes in Computer Science vol. 4116, pp. 156–172, Springer–Verlag, 2006.

[LAMA07] LaMacchia B., Lauter K., Mityagin A.: Stronger Security of Authenticated Key Exchange. Lecture Notes in Computer Science, vol. 4784, pp. 1–16, Springer–Verlag, 2007.

[LAMB07] Lambert R., Vadekar A.: Method and Apparatus for Computing a Shared Key. US Patent 7512233.

[LAW03] Law L., Menezes A., Qu M., Solinas J., Vanstone S.: An Efficient protocol for authenticated key agreement. "Designs, Codes and Cryptography", vol. 28(2), pp 119–134, Kluwer Academic Publishers, 2003.

[LEE08a] J. Lee, C. S. Park. *An Efficient Authenticated Key Exchange Protocol with a Tight Security Reduction.* Cryptology ePrint Archive, Report 2008/345, 2008.

[LEE08b] Lee J., Park J. H.: Authenticated key exchange secure under the computational Diffe–Hellman assumption. Cryptology ePrint Archive, Report 2008/344, 2008.

[LEN87] Lenstra H. W.: Factoring Integers with Elliptic Curves. Annals of Mathematics, vol. 126, pp. 649–673, 1987.

[LOP99] López J., Dahab R.: Fast Multiplication on Elliptic Curves Over $GF(2^m)$ without precomputation. In proc. of Cryptographic Hardware and Embedded Systems — CHES 99, Lecture Notes in Computer Science, vol. 1717, pp. 316–327, Springer–Verlag, 1999.

[MAU96] Maurer U. M., Wolf S.: Diffie–Hellman Oracles. Lecture Notes in Computer Science, vol. 1109, pp. 268–282, Springer–Verlag, 1996.

[MEN93] Menezes A., Okamoto T., S. Vanstone.: Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Transactions on Information Theory, vol 39(5), pp. 1639–1646, 1993.

[MEN96] Menezes A., van Oorschot P., Vanstone S.: Handbook of Applied Cryptography. CRC Press, 1996.

[MEN01] Menezes A., Qu M.: Analysis of the Weil Descent Attack of Gaudry, Hess and Smart. In Proc. of Topics in Cryptology — CT–RSA 2001, Lecture Notes in Computer Science, vol. 2020, pp. 308–318, Springer–Verlag, 2001.

[MEN06] Menezes A., Ustaoglu B.: On the Importance of Public–Key Validation in the MQV and HMQV Key Agreement Protocols. In Proc. of Indocrypt 2006, Lecture Notes in Computer Science, vol. 4329, pp. 133-147, Springer–Verlag, 2006.

[MEN07] Menezes A.: Another Look at HMQV. Journal of Mathematical Cryptology, vol. 1, pp. 148–175, Walter de Gruyter, 2007.

[MEN08] Menezes A., Ustaoglu B.: Security Arguments for the UM Key Agreement Protocol in the NIST SP 800-56A Standard. In Proc. of of ACM Symposium on Information, Computer and Communications Security 2008, pp. 261–270, ACM, 2008.

[MEN09] Menezes A., Ustaoglu B.: Comparing the Pre– and Post–specified Peer Models for Key Agreement. International Journal of Applied Cryptography, vol. 1(3) pp. 236–250, 2009 (available from http://www.cacr.math.uwaterloo.ca/~ajmeneze).

[MER09] Merli C.: Lite Security Module. at http://www.clizio.com/lsmpkcs11.html, 2009.

[MIL86] Miller V. S.: Use of Elliptic Curves in Cryptography. In Proc of Crypto 85, Lecture Notes in Computer Science, vol. 218, pp. 417-426, Springer–Verlag, 1986.

[MIY97] Miyaji A., Ono T., Cohen H.: Efficient elliptic curve exponentiation. Information and Communication Security — ICICS 1997, Lecture Notes in Computer Science, vol. 1334, pp. 282–290, Springer–Verlag, 1997.

[MON87] Montgomery P.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Mathematics of computation, vol. 48(177), pp. 243–264, AMS, 1987.

[MOR90] Morain F., Olivos J.: Speeding up the computations on an elliptic curve using addition-subtraction chains. Informatique Théorique et Applications, vol. 24, pp. 531–543, 1990.

[MOZ09] Mozilla Foundation: Common PKCS #11 Implementation Problems (Revision 48936), at http://www.mozilla.org/projects/security/pki/pkcs11/netscape/problems.html, 2009.

[MRA96] M'Raïhi D., Naccache D.: Batch Exponentiation: A Fast DLP-based Signature Generation Strategy. In Proc. of the third ACM conference on Computer and communications security, pp. 58–61, ACM, 1996.

[NIS03] NIST Special publication 800–56: Recommendation on Key Establishment Schemes, 2003.

[NIS07] NIST Special publication 800–56A: Recommendation for Pair–Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, 2008.

[OKA01] Okamoto T., Pointcheval D.: The Gap–Problems: A New Class of Problems for the Security of Cryptographic Schemes. In Proc. of Crypto 96, Lecture Notes in Computer Science, vol. 1992, pp. 104–118, Springer–Verlag, 2001.

[OKE01] Okeya K., Sakurai K.: Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve. In Proc of Cryptographic Hardware and Embedded Systems — CHES 2001, Lecture Notes in Computer Science, vol. 2162, pp. 126–141, 2001.

[PHI95] Philips Semiconductors: The I2C–bus and how to use it (including specifications) (available from http://www.mcc-us.com/i2cHowToUseIt1995.pdf), 1995.

[POH78] Pohlig S., Hellman M.: An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. IEEE Transactions on Information Theory, vol. 24, pp. 106–110, 1978.

[POI00] Pointcheval D., Stern J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology, vol. 13, pp. 361–396, Springer–Verlag, 2000.

[POL00] Pollard J. M.: Kangaroos, Monopoly and Discrete Logarithms. Journal of Cryptology, vol. 13, pp. 437–447, 2000.

[POL78] Pollard J. M.: Monte Carlo methods for index computation mod p. Mathematics of computations,vol. 32, pp. 918–924, AMS, 1978.

[QUI01] Quisquater J.-J., Samyde D.: Electromagnetic analysis (EMA): Measures and counter–measures for smart cards. In Proc. of Smart

Card Programming and Security 2001, Lecture Notes in Computer Science, vol. 2140, pp. 200–210. Springer–Heidelberg, 2001.

[REI60] Reitwiesner G. W.: Binary arithmetic. Advances in computers, vol. 1, pp. 231–308, Academic Press, 1960.

[RSA04] RSA Lab.: PKCS #11 v2.20: Cryptographic Token Interface Standard. RSA Laboratories, 2004.

[SAR08] Sarr A. P.: Analysis of the ECMQV Protocol. Technical report, 2008.

[SAR09a] Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A Secure and Efficient Authenticated Diffie–Hellman Protocol. In Proc. of Public Key Infrastructures, Services and Applications — EuroPKI 2009, Lecture Notes in Computer Science, vol. 6391, pp. 83–98, Springer–Verlag, 2010.

[SAR09b] Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A Secure and Efficient Authenticated Diffie–Hellman Protocol (extended version). Cryptology ePrint Archive, Report 2009/408, 2009.

[SAR10a] Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: Enhanced Security and Efficiency for Authenticated Key Agreement. International Workshop on Foundations of Security and Privacy — FCS-PrivMod 2010.

[SAR10b] Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A New Security Model for Authenticated Key Agreement (extended version). Cryptology ePrint Archive, Report 2010/237, 2010.

[SAR10c] Sarr A. P., Elbaz–Vincent Ph., Bajard J. C.: A New Security Model for Authenticated Key Agreement. In Proc. of the 7th International Conference on Security and Cryptography for Networks — SCN 2010, Lecture Notes in Computer Science, vol. 6280, pp. 219–234, Springer–Verlag, 2010.

[SAT00] Satoh T.: The Canonical Lift of an Ordinary Elliptic Curve over a Finite Field and its Point Counting. Journal of the Ramanujan Mathematical Society, vol. 15(4), pp. 247–270, 2000.

[SAT98] Satoh T., Araki K.: Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. Commentarii Mathematici Universitatis Sancti Pauli, vol. 47, pp. 81–92, 1998.

[SHA71] Shanks D.: Class number, a theory of factorization, and genera. In Proc. of Symposia in Pure Mathematics, vol. 20, pp. 415–440, AMS, 1971.

[SCH91] Schnorr C. P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology, vol. 4(3), pp. 161–174, Springer New York, 1991.

[SCH85] Schoof R.: Elliptic curves over finite fields and the computation of square roots mod p. Mathematics of computation, vol. 44(170), pp. 483–494, 1985.

[SCH95] Schoof R.: R. Schoof. Counting points on elliptic curves over finite fields. Journal de Théorie des Nombres de Bordeaux, vol. 7(1), pp. 219–254, 1995.

[SEM98] Semaev I.: Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. Mathematics of Computation, vol. 67, pp. 353–356, 1998.

[SHO99] Shoup V.: On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, 1999/012, 1999.

[SIL86] Silverman J. H.: The Arithmetic of Elliptic Curves. Number 106 in Graduate Texts in Mathematics, Springer Verlag, 1986.

[SIL92] Silverman J. H., Tate J.: Rational Points on Elliptic Curves. Undergraduate Texts in Mathematics, Springer Verlag, 1992.

[SKJ03] Skjernaa. B.: Satoh's algorithm in characteristic 2. Mathematics of Computation, vol. 72(241), pp. 477–487, 2003.

[SMA99] Smart N.: The discrete logarithm problem on elliptic curves of trace one. Journal of Cryptology, vol. 12, pp. 193–196, 1999.

[STI95] Stinson D. R.: Cryptography: Theory and Practice. CRC Press, 1995.

[SUN06] SUN Corp.: Java™ Cryptography Architecture (JCA) Reference Guide, 2006.

[TES01a] Teske E.: On random walks for Pollard's rho method. Mathematics of Computation, vol. 70, pp. 809–825, AMS, 2001.

[TES01b] Teske E.: Square-root Algorithms for the Discrete Logarithm Problem (A survey). Public Key Cryptography and Computational Number Theory, pp. 283–301, Walter de Gruyter, 2001.

[ULT09] Ultimaco: SafeGuard CryptoServer Hardware Security Modules Datasheet. 2009 (available from http://hsm.utimaco.com/fileadmin/assets/Leaflet/DS_SGCS_CryptoServer_en.pdf.)

[UST08] Ustaoglu B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. "Designs, Codes and Cryptography", vol. 46(3), pp. 329–342, ACM, 2008.

[UST09] Ustaoglu B.: Comparing *SessionStateReveal* and *EphemeralKeyReveal* for Diffie-Hellman protocols. In proceedings of ProvSec 2009, Lecture Notes in Computer Science, vol. 5848, pp. 183–197, Springer–Verlag, 2009.

[VAN99] Van Oorschot P. C., Wiener M. J.: Parallel Collision Search with Cryptanalytic Applications. Journal of Cryptology, vol. 12(1), pp. 1–28, 1999.

[VAN92] Vanstone S.: Responses to NIST's Proposal. Communications of the ACM, vol. 35, 50-52, ADCM, 1992.

[WAS08] Washington L.: Elliptic Curves: Number Theory and Cryptography(Second Edition). CRC Press, 2008.

[WEI08] Weingart S. H.: Physical Security Devices for Computer Subsystems : A Survey of Attacks and Defenses. (An update of the version published at CHES 2000, available from http://www.atsec.com/downloads/pdf/phy_sec_dev.pdf), 2008.

[WIE99] Wiener M. J., Zuccherato R. J.: Faster Attacks on Elliptic Curve Cryptosystems. Lecture Notes in Computer Science, vol. 1556, pp. 190–200, Springer–Verlag, 1999.

[WON01] Wong D. S., Chan A. H.: Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices. Lecture Notes in Computer Science, vol. 2248, pp. 172–289, Springer–Verlag, 2001.

Companies and Organizations

[ARM] ARM: http://www.arm.com.

[BUL] Bull: http://www.bull.fr.

[CER] Certicom: http://www.certicom.com

[EVE] Everbee: http://www.everbee.com.

[GEM] Gemalto: http://www.gemalto.com.

[IBM] IBM Security: http://www-03.ibm.com/security.

[MAX] Maxim: http://www.maxim-ic.com.

[MOZ] Mozilla Foundation http://www.mozilla.org.

[NET] Netheos R&D: http://www.netheos.net.

[RSAL] RSA Laboratories: http://www.rsa.com/rsalabs.

[SCM] SCM Microsystems: http://www.scmmicro.com.

[THA] Thales Security: http://www.thalesgroup.com/security.

[ULT] Utimaco: http://www.utimaco.de.

[UPEK] UPEK: http://www.upek.com.

Index

Abstract. An impressive ratio of the previously proposed key agreement protocols turn out to be insecure when regarded with respect to recent security models. The Canetti–Krawczyk (CK) and extended Canetti–Krawczyk (eCK) security models, are widely used to provide security arguments for key agreement protocols. We point out security shades in the (e)CK models, and some practical attacks unconsidered in (e)CK–security arguments. We propose a strong security model which encompasses the eCK one. We propose a complementary analysis of the Exponential Challenge Response (XRC) and Dual Exponential Challenge Response (DCR) signature schemes, which are the building blocks of the HMQV protocol. On the basis of this analysis we show how impersonation and man in the middle attacks can be performed against the (C, H)MQV(–C) protocols when some session specific information leakages happen. We define the Full Exponential Challenge Response (FXRC) and Full Dual Exponential Challenge Response (FDCR) signature schemes; using these schemes we propose the Fully Hashed MQV protocol and the Strengthened MQV protocol, which preserve the remarkable performance of the (H)MQV protocols and resist the attacks we present. The SMQV and FHMQV protocols are particularly suited for distributed implementations wherein a tamper–proof device is used to store long–lived keys, while session keys are used on an untrusted host machine. In such settings, the non–idle time computation effort of the device reduces to few non–costly operations. The SMQV and FHMQV protocols meet our security definition under the Gap Diffie–Hellman assumption and the Random Oracle model.

Key words: authenticated key agreement, practical vulnerability, CK model, eCK model, FXCR scheme, FDCR scheme, strengthened eCK model, MQV, HMQV, FHMQV, SMQV.

Résumé. Une part importante des protocoles d'échange de clefs proposés se sont révélés vulnérables lorsqu'analysés au regard des définitions de sécurité les plus récentes. Les arguments de sécurité des protocoles récents sont généralement fournis avec les modèles de sécurités dits de Canetti–Krawczyk (CK) et Canetti–Krawczyk étendus (eCK). Nous montrons que ces définitions de sécurité présentent des subtilités qui font que certaines attaques, qui peuvent être menées en pratique, ne sont pas considérées dans les analyses de sécurité. Nous proposons une forte définition de sécurité, qui englobe le modèle eCK. Nous proposons une analyse complémentaire des schémas de signature XCR ("Exponential Challenge Response") et DCR ("Dual exponential Challenge Response"), qui sont les briques du protocole HMQV. Sur la base de cette analyse, nous montrons la vulnérabilités des protocoles (C, H)MQV(–C) aux fuites d'informations spécifiques à une session. Nous montrons notamment que lorsqu'un attaquant accède à certaines informations de session, qui ne conduisent pas à une divulgation de la clef statique du détenteur de la session, il peut réussir une attaque par usurpation d'identité. Nous proposons les schémas de signature FXCR ("Full XCR") et FDCR ("Full DCR") à partir desquels nous construisons les protocoles FHMQV ("Fully Hashed MQV") et SMQV ("Strengthened MQV") qui préservent la performance remarquable des protocole (H)MQV, en plus d'une meilleure résistance aux fuites d'informations. Les protocoles FHMQV et SMQV sont particulièrement adaptés aux environnements dans lesquels une machine non digne de confiance est combinée avec un module matériel à faible capacité de calcul et résistant aux violations de sécurité. Dans un tel environnement, les opérations effectuées sur le module matériel hors temps mort se réduisent à des opérations peu coûteuses. Les protocoles FHMQV et SMQV satisfont notre définition de sécurité sous les hypothèses de l'oracle aléatoire et du problème échelon de Diffie-Hellman.

Mots clefs: protocoles d'échange de clefs authentifiés, vulnérabilités, modèle CK, modèle eCK, schema de signature FXCR, schema de signature FDCR, modèle eCK fortifié, MQV, HMQV, FHMQV, SMQV.