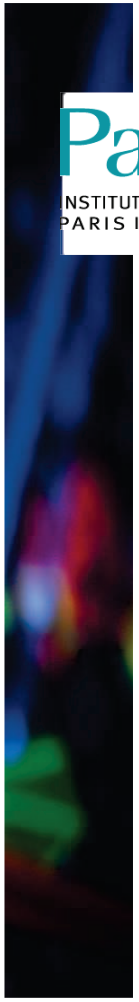
Physical security of elliptic curve cryptography

Cédric Murdica



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2014-ENST-0008

EDITE ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

Télécom ParisTech

Spécialité “ Électronique et Communications ”

présentée et soutenue publiquement par

Cédric MURDICA

le 13 février 2014

Sécurité Physique de la Cryptographie sur Courbes Elliptiques

Directeurs de thèse : **Philippe HOOGVORST, David NACCACHE**
Co-encadrement de la thèse : **Jean-Luc DANGER**

Jury

M. Pierre-Alain FOUQUE, Professeur, Université de Rennes I	Président, Rapporteur
M. Jean-Sébastien CORON, Maître de Conférences, Université du Luxembourg	Rapporteur
M. Jean-Jacques QUISQUATER, Professeur, Université Catholique de Louvain	Examinateur
M. Philippe NGUYEN, Directeur Technique, Secure-IC	Examinateur
M. Sylvain GUILLEY, Professeur Associé, Télécom ParisTech	Invité

T H È S E

Télécom ParisTech
école de l’Institut Mines Télécom – membre de ParisTech
46, rue Barrault – 75634 Paris Cedex 13 – Tél. + 33 (0)1 45 81 77 77 – www.telecom-paristech.fr

Abstract

Elliptic Curve Cryptography (ECC) has gained much importance in smart cards because of its higher speed and lower memory needs compared with other asymmetric cryptosystems such as RSA. ECC is believed to be unbreakable in the black box model, where the cryptanalyst has access to inputs and outputs only. However, it is not enough if the cryptosystem is embedded on a device that is physically accessible to potential attackers. In addition to inputs and outputs, the attacker can study the physical behaviour of the device. This new kind of cryptanalysis is called *Physical Cryptanalysis* where two main families arise: *Side-Channel* and *Fault Attacks*. Side-Channel Attacks exploit information leaking during the execution of a cryptographic algorithm embedded in a device. In a Fault Attack, the attacker forces the device into an abnormal mode of operation. The attacker can potentially derive the secrets stored in the system from the wrong results. This thesis focuses on physical cryptanalysis of ECC.

The first part gives the background on ECC. From the lowest to the highest level, ECC involves a hierarchy of tools: Finite Field Arithmetic, Elliptic Curve Arithmetic, Elliptic Curve Scalar Multiplication and Cryptographic Protocol. Depending on the physical attack, the cryptanalyst must have a certain knowledge of the implementation to a certain level. Therefore, each level of the hierarchy is described in detail in a chapter.

The second part exhibits a state-of-the-art of the different physical attacks and countermeasures on ECC. For each attack, the context on which it can be applied is given while, for each countermeasure, we estimate the time and memory cost. We propose new attacks and new countermeasures. Then, we give a clear synthesis of the attacks depending on the context. This is useful during the task of selecting the countermeasures. Finally, we give a clear synthesis of the efficiency of each countermeasure against the attacks.

Résumé

La Cryptographie sur les Courbes Elliptiques (abréviée ECC de l'anglais *Elliptic Curve Cryptography*) est devenue très importante dans les cartes à puces car elle présente de meilleures performances en temps et en mémoire comparée à d'autres cryptosystèmes asymétriques comme RSA. ECC est présumé incassable dans le modèle dit "Boîte Noire", où le cryptanalyste a uniquement accès aux entrées et aux sorties. Cependant, ce n'est pas suffisant si le cryptosystème est embarqué dans un appareil qui est physiquement accessible à de potentiels attaquants. En plus des entrés et des sorties, l'attaquant peut étudier le comportement physique de l'appareil. Ce nouveau type de cryptanalyse est appelé *cryptanalyse physique*, qui se distinguent en deux grandes familles: *attaques par canal auxiliaire* et *attaques en fautes*. Les attaques par canal auxiliaire exploitent l'information émanant de l'appareil pendant l'exécution d'un algorithme cryptographique. Concernant les attaques en fautes, l'attaquant force l'appareil à effectuer un mode d'opérations anormal. À partir des faux résultats, l'attaquant peut potentiellement dériver les secrets stockés sur la carte. Cette thèse porte sur les attaques physiques sur ECC.

La première partie fournit les pré-requis sur ECC. Du niveau le plus bas au plus élevé, ECC nécessite les outils suivants : l'arithmétique sur les corps finis, l'arithmétique sur courbes elliptiques, la multiplication scalaire sur courbes elliptiques et enfin les protocoles cryptographiques. Les attaques physiques nécessitent une certaine connaissance de l'implémentation visée jusqu'à un certain niveau dans la hiérarchie. Ainsi, chaque niveau est décrit de façon détaillée dans un chapitre.

La deuxième partie expose un état de l'art des différentes attaques physiques et contremesures sur ECC. Pour chaque attaque, nous donnons le contexte dans lequel elle est applicable. Pour chaque contremesure, nous estimons son coût en temps et en mémoire. Nous proposons de nouvelles attaques et de nouvelles contremesures. Ensuite, nous donnons une synthèse claire des attaques suivant le contexte. Cette synthèse est utile pendant la tâche du choix des contremesures. Enfin, une synthèse claire de l'efficacité de chaque contremesure sur les attaques est donnée.

Remerciements

Trois années de thèse s'achèvent. C'est une expérience très enrichissante, mais aussi difficile. Il est donc normal de remercier les gens qui m'ont aidé de près ou de loin, de quelque façon que ce soit.

Tout d'abord je tiens à remercier Philippe Hoogvorst et David Naccache pour avoir rempli le rôle de directeurs de thèse. Leurs conseils étaient toujours très utiles. Un grand merci également à Jean-Luc Danger et Sylvain Guilley, ainsi qu'à tout le département COMELEC, qui ont initié et encadré cette thèse.

Je remercie les fondateurs de Secure-IC : Hassan Triqui, Philippe Nguyen, Sylvain Guilley, Jean-Luc Danger et Laurent Sauvage de m'avoir accepté au sein de l'entreprise. À mon arrivée, la spin-off n'existait que depuis dix mois et comptait très peu d'employés. C'est un honneur d'avoir intégré l'équipe à ses débuts.

J'ai eu la chance d'avoir un jury prestigieux: Jean-Sébastien Coron, Pierre-Alain Fouque, Jean-Jacques Quisquater, Philippe Nguyen, Jean-Luc Danger et Sylvain Guilley. Je leur remercie d'avoir accepter d'évaluer mes travaux.

Le corps administratif de Télécom ParisTech a été très efficace. Je cite notamment Bruno Thedrez, Alain Sibille, Florence Besnard, Chantal Cadiat, Zouina Sahnoune et Fabienne Lassausaie. Leur présence et leur gentillesse m'ont beaucoup facilité les choses.

Merci à tous mes co-auteurs qui ont permis de produire des articles de grande qualité: mes encadrants Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, et David Naccache; et aussi Diana Maimuţ et Mehdi Tibouchi qui ont largement contribué à l'article sur l'attaque en faute.

Je tiens aussi à remercier tous mes collègues de Secure-IC, anciens et actuels. Ce sont finalement les personnes que je côtoie le plus souvent. Par ordre de rencontre : Sébastien B, Alexandre C, Matthieu L, Lionel T, Molka B, Karine L, Alejandro L, Thibault P, Anne-Sophie D, Pierre V, Clément L, Romain H, Yann B, Olivier P, Robert N, Charles T, Olivier E, Brice M, Chloé F, Théophile B, Janie Q, Rachid D, Youssef S, Cécile P, Valentin P et François R.

Ce fût trois années joyeuses notamment grâce aux nombreuses soirées au O'Connell, au Comptoir ou au Frogs. C'est aussi grâce au Baby-foot qui anime la coupure du midi. Oui, nous avons un Baby-foot à Secure-IC!

Enfin, et non le moins important, je remercie mes parents, mes frères, mes belles sœurs, et ma nièce d'avoir toujours cru en moi. Leur soutien était indispensable. J'en profite pour

remercier aussi tous mes amis. Ils ont aussi joué leur rôle dans cette thèse. Malgré des choix professionnels complètement différents, leurs conseils et leur soutien se sont faits ressentir. Tout s'est fait dans la joie et la bonne humeur, principalement au Belise, au Coq ou à la Duchesse Anne.

Description des travaux

La cryptographie est une science qui permet de protéger des messages. Avant l'envoi d'un message, il est d'abord transformé de façon à ce qu'il soit incompréhensible sauf pour le destinataire du message : c'est le chiffrement. La confidentialité est alors assurée. La méthode inverse est appelé le déchiffrement. En plus de la confidentialité, la cryptographie remplit d'autres fonctionnalités telles que l'authentification et l'intégrité.

Par le principe de Kerckhoffs, la sécurité d'un cryptosystème doit uniquement reposer sur une donnée secrète et non sur les méthodes utilisées pour chiffrer ou déchiffrer des messages. Cette donnée secrète est appelée *clé*. Deux grandes familles de cryptosystèmes existent: la cryptographie symétrique et asymétrique.

Dans un cryptosystème symétrique, la même clé est utilisée pour chiffrer et déchiffrer des messages.

Dans un cryptosystème asymétrique, deux clés sont utilisées. Une clé est publiquement diffusée et sert à effectuer des procédures publiques telles que le chiffrement de messages ou la vérification de signatures. Cette clé est appelée *clé publique*. Lorsqu'un message est chiffré, personne n'est capable de le déchiffrer sauf le propriétaire de la seconde clé, appelée *clé privée*. Le détenteur de cette clé privée est aussi le seul capable de signer des messages. La cryptographie asymétrique apporte plus de fonctionnalités que la cryptographie symétrique mais elle nécessite des calculs beaucoup plus importants que la cryptographie symétrique. Généralement, la cryptographie asymétrique est utilisée au départ d'une communication entre deux entités pour l'authentification et l'échange d'une clé symétrique. Une fois fait, les deux entités communiquent en utilisant un cryptosystème symétrique avec la clé échangée que seules ces entités connaissent. RSA fut le premier cryptosystème asymétrique, introduit par Rivest, Shamir et Adleman en 1977. À la fin des années 1980, Koblitz et Miller ont présenté l'utilisation des courbes elliptiques pour des applications cryptographiques. Cette thèse se focalise sur la Cryptographie sur Courbes Elliptiques (abréviée ECC, de l'anglais *Elliptic Curve Cryptography*).

La sécurité d'un cryptosystème est assurée par de fortes preuves mathématiques dans le modèle de la boîte noire. Dans ce modèle, l'attaquant a uniquement accès aux entrées et aux sorties.

La cryptographie est beaucoup utilisée dans les cartes à puces. Un nouveau type d'attaques sur les cartes à puces a vu le jour à la fin des années 1990. Kocher a montré qu'une simple analyse du temps d'exécution était suffisante pour récupérer la clé utilisée dans la carte ciblée. Depuis, de nombreuses attaques de ce type ont émergé. Elles se basent sur l'observation du comportement de la cible pendant l'exécution d'un algorithme cryptographique. Ce type d'attaques est appelé *attaques physiques*. Les preuves de sécurité dans le modèle de la boîte noire ne sont pas suffisantes dans ce cas. Bien sûr, de nombreuses méthodes existent pour contrer ces attaques.

Cette thèse s'adressent aux designers développant des applications cryptographiques à base de courbes elliptiques embarquées sécurisées. La première partie donne les pré-requis de ECC. La seconde partie se focalisent sur les attaques physiques et contremesures sur ECC. En plus d'un état-de-l'art complet, nous introduisons de nouvelles attaques et de nouvelles contremesures.

Partie I : Cryptographie sur Courbes Elliptiques

Cette partie fournit les pré-requis sur ECC. Du niveau le plus bas au plus élevé, ECC nécessite les outils suivants : l'arithmétique sur les corps finis, l'arithmétique sur courbes elliptiques, la multiplication scalaire sur courbes elliptiques et enfin les protocoles cryptographiques. Les attaques physiques nécessitent une certaine connaissance de l'implémentation visée jusqu'à un certain niveau dans la hiérarchie. Ainsi, chaque niveau est décrit de façon détaillée dans un chapitre.

Cette partie est en fait un état-de-l'art des différentes méthodes de calcul sur les courbes elliptiques.

Chapitre 1 : Définition des Courbes Elliptiques

Ce chapitre décrit les courbes elliptiques et leurs propriétés. À savoir, une courbe elliptique sur un corps $\mathbb{K}$ de caractéristique différente de deux et trois, est défini par son équation de Weierstraß réduite :

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

avec $a, b \in \mathbb{K}$ vérifiant $4a^3 + 27b^2 \neq 0$.

Dans ce chapitre, nous donnons également les propriétés principales des courbes elliptiques. Parmi celles-ci, nous insistons sur leur structure de groupe car c'est ce qui fait des courbes elliptiques de bons outils pour la cryptographie. Les points de la courbe elliptiques forment un groupe abélien additif, en suivant la règle de la sécante tangente illustré ci-dessous sur $\mathbb{R}$.

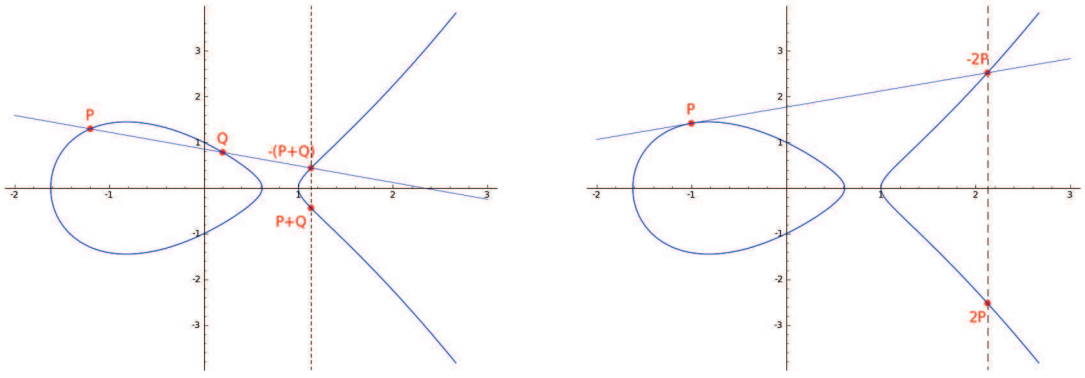


Figure 1: Addition et doublement de points sur la courbe $y^2 = x^3 - 2x + 1$ sur $\mathbb{R}$

Nous présentons également le système de coordonnées projectives. Dans ce système, les points de la courbe sont définis avec trois coordonnées (X, Y, Z) au lieu de deux (x, y). Ce système et ses variantes (tel que les coordonnées Jacobiennes) sont très utilisés car ils permettent d'éviter d'effectuer de nombreuses inversions modulaires très coûteuses.

Chapitre 2 : Arithmétique sur les Corps Finis

Les courbes elliptiques les plus utilisées sont les courbes elliptiques définies sur un corps premier de grande caractéristique. Ainsi, les opérations de bases sur les courbes elliptiques, mais aussi sur RSA, sont les opérations sur le corps $\mathbb{F}_p$ avec $p > 3$. Ces opérations sont très importantes car elles doivent être très performantes. Elles sont donc très étudiées tant au niveau algorithmique qu'au niveau de l'implémentation logicielle ou matérielle.

Certaines attaques physiques se focalisent sur ce niveau le plus bas de la hiérarchie. Nous décrivons donc en détails certaines implémentations pour une grande compréhension de ces attaques. Nous décrivons notamment en détails la multiplication de Montgomery qui très largement répandue.

Chapitre 3: Arithmétique sur les Courbes Elliptiques

De nombreuses formules existent pour calculer l'addition de deux points ou le doublement d'un point. On trouve aussi d'autres formules plus exotiques telles que l'addition conjugué de deux points P et Q qui calcule $P + Q$ et $P - Q$ au sein de la même formule.

Ce chapitre synthétise les différentes formules les plus utilisées. Une synthèse sur le coût des formules en temps et en mémoire est fourni à la fin de ce chapitre.

Chapitre 4 : Multiplication Scalaire

La multiplication scalaire sur courbes elliptiques (ECSM de l'anglais *Elliptic Curve Scalar Multiplication*) est l'opération qui consiste à calculer

$$[k]P = \underbrace{P + \cdots + P}_{k \text{ fois}},$$

à partir d'un point P de la courbe donnée et d'un entier k.

Ce chapitre donne les algorithmes de multiplication scalaire les plus utilisés. Les multiplications scalaires sont répertoriées en fonction de la régularité. Un ECSM est dit *régulier* si, à chaque itération, les mêmes opérations sur la courbe sont effectués quel que soit la valeur du scalaire.

Cette distinction est importante pour la sécurité physique puisqu'une multiplication scalaire régulière est protégée face à l'une des premières attaques en canaux auxiliaires : l'attaque Simple Side-Channel Analysis.

De même que pour les formules sur courbes elliptiques, une synthèse est donnée à la fin du chapitre sur le coût des différents algorithmes de multiplication scalaire.

Chapitre 5 : Protocoles Cryptographiques

Ce chapitre décrit certains protocoles cryptographiques basés sur les courbes elliptiques. Le protocole de signature ECDSA, le protocole d'échange de clés ECDH et le protocole de chiffrement EC-ELGAMAL sont décrits. Voici la procédure de signature ECDSA qui sera utile par la suite.

Algorithm 1 ECDSA Signature

Entrée: clé privée d, un entier encodé $m \in [0, p-1]$ représentant un message
Sortie: Signature (r, s)

1: $k \xleftarrow{\mathcal{R}} \{1, \ldots, t-1\}$
2: $Q \leftarrow [k]G$
3: $r \leftarrow x_Q \bmod t$
4: **if** $r = 0$ **then**
5: **go to** ligne 1
6: $k_{inv} \leftarrow k^{-1} \bmod t$
7: $s \leftarrow k_{inv}(dr + m) \bmod t$
8: **if** $s = 0$ **then**
9: **go to** ligne 1
10: **return** (r, s)

Toutes les procédures privées de ces protocoles nécessitent le calcul d'une multiplication scalaire $[k]P$ avec k devant rester impérativement secret sinon la clé privée du système cryptographique est retrouvée.

Chapitre 6 : Sécurité de ECC

La sécurité de ECC dans le modèle de la boîte noire repose sur la difficulté du **Logarithme Discret sur Courbes Elliptiques** (ECDLP de l'anglais *Elliptic Curve Discrete Logarithm Problem*) ou de ses variantes. Le problème ECDLP consiste à retrouver k en ayant accès à P et $Q = [k]P$.

Ce problème est considéré comme difficile. Actuellement, les meilleurs algorithmes permettant de le résoudre sont l'algorithme rho de Pollard [Pol78] et l'algorithme Baby-step Giant-Step [Sha71]. Ils ont tous les deux une complexité de $\mathcal{O}(\sqrt{t})$ où $t = ord(P)$. Lors d'une application cryptographique, si l est le paramètre de sécurité (par exemple 128 ou 256), la courbe elliptique est choisie de telle sorte qu'il existe un point P d'ordre $ord(P) \approx 2^{2l}$.

Généralement, les attaques physiques visent à récupérer le scalaire k avec des méthodes totalement différentes, sans résoudre le ECDLP. Heureusement, il existe aussi des méthodes pour parer ces attaques. C'est le sujet de la seconde partie de la thèse.

Partie II : Attaques physiques et contremesures sur ECC

Cette partie décrit les différentes attaques physiques et contremesures sur ECC. Les attaques et contremesures sont très nombreuses et doivent être décrites suivant une méthodologie. Le chapitre 7 explique comment les différentes attaques seront décrites. Le chapitre 8 est le cœur de la thèse car c'est dans ce chapitre qu'on liste toutes les attaques et contremesures. Nous présentons également de nouvelles attaques et contremesures. Dans le chapitre 9, nous expliquons comment il est possible d'effectuer des attaques en faute différentielles sur ECDSA. Enfin, les chapitres 10 et 11 sont des synthèses des attaques et des contremesures.

Chapitre 7 : Caractérisation

Étant donné le très grand nombre d'attaques physiques sur ECC, il est important de choisir des termes précis pour les décrire. Les attaques sont classées en trois catégories :

- **Attaques par canaux auxiliaires.** L'attaquant observe le comportement de la cible sans le perturber. Jusqu'à maintenant, les différents attaques par canaux auxiliaires sont:
  - **Attaques temporelles.** L'attaquant déduit de l'information sur le secret en analysant le temps d'exécution.
  - **Attaques par analyse simple.** L'attaquant observe les différents motifs de la consommation courant ou du rayonnement électromagnétique pendant l'exécution d'un algorithme cryptographique.
  - **Attaques par templates.** L'attaquant contrôle une carte (c'est à dire qu'il peut choisir les données mais aussi les secrets) ayant exactement les mêmes caractéristiques physiques que la carte ciblée. Il récupère la trace de consommation courant ou du rayonnement électromagnétique de sa carte qu'il contrôle en faisant varier des sous-parties de la clé. Cette phase constitue les templates. Il récupère ensuite la trace de la cible et compare celle-ci avec les templates pour conclure quel secret est le plus probable.
  - **Attaques en canaux auxiliaires verticales.** Plusieurs ECSMs sont exécutés et l'attaquant récupère la trace de consommation à chaque fois. Un outil statistique est appliqué sur les traces pour déduire les valeurs utilisées et donc le secret.
  - **Attaques en canaux auxiliaires horizontales.** Une seule trace est disponible. L'attaquant utilise des outils statistiques sur des segments de la trace.
- **Attaques en faute.** L'attaquant perturbe le système et les résultats éventuellement incorrects peuvent déduire de l'information. Pour ECC, les différentes attaques en faute sont :
  - **Attaques Safe-Error.** L'attaquant injecte une faute sur une zone précise à un moment précis de l'exécution. Le résultat final sera incorrect uniquement si le secret vérifie certaines conditions.
  - **Attaques par courbes faibles.** Une donnée est perturbée. L'ECSM est effectué sur une courbe qui est plus faible que la courbe de départ.
  - **Attaques en fautes différentielles.** Plusieurs ECSMs sont exécutés. Á chaque fois, une faute est introduite. Les résultats incorrectes sont comparés avec les bons résultats ou entre eux pour déduire de l'information sur le secret.
- **Attaques combinées.** L'attaquant peut combiner deux attaques ou plus, éventuellement une attaque par canaux auxiliaires et en faute.

Le contexte de chaque attaque sera aussi décrit. En effet, certaines attaques ne fonctionnent que sur certaines implémentations, ou certaines courbes, ou si l'attaquant peut choisir le point de base de l'ECSM. Pour chaque attaque, nous donnons les informations suivantes:

- **Récupération de la clé.** Une description de la procédure de récupération de la clé est donnée.
- **Particularité de la courbe elliptique.** Nous indiquons si l'attaque fonctionne uniquement sur certaines courbes.

- **Particularité de l'implémentation.** Nous indiquons si l'attaque fonctionne uniquement sur des implémentations particulières.
- **Nombre d'exécutions.** Le nombre d'exécutions nécessaire pour retrouver l'intégralité du scalaire est donné.
- **Accès au point de base.** Nous indiquons si l'attaquant doit choisir le point de départ, ou s'il doit le connaître ou s'il n'a aucune importance.
- **Accès au résultat.** Nous indiquons si l'attaquant doit connaître ou non le résultat de l'ECSM.
- **Modèle de faute.** Pour les attaques en faute, nous indiquons la précision de la faute nécessaire pour que l'attaque puisse fonctionner.

Chaque contremesure a un certain coût. Nous précisons le coût en utilisant les notations suivantes :

- $\text{ECSM}_{l,n}$: temps d'exécution d'un ECSM avec un scalaire de l bits et un module de n bits,
- $\text{ECADD}_n, \text{ECDBL}_n, \text{C-ECADD}_n$: temps d'exécution d'une addition, d'un doublement et d'une addition conjuguée respectivement avec un module de n bits,
- $\text{ADD}_n, \text{SQR}_n, \text{MUL}_n, \text{DIV}_n$: temps d'exécution d'une addition/soustraction, d'un carré, d'une multiplication et d'une division respectivement, avec des entiers de n bits,
- $\text{mADD}_n, \text{mSQR}_n, \text{mMUL}_n, \text{mINV}_n$: temps d'exécution d'une addition/soustraction modulaire, d'un carré modulaire, d'une multiplication modulaire et d'une inversion modulaire respectivement, avec des entiers de n bits,
- RNG_n: temps d'exécution de la génération d'un nombre aléatoire de n bits,
- RPG_m: temps d'exécution de la génération d'une permutation aléatoire de m éléments,
- CRC_n: temps d'exécution d'un contrôle de redondance cyclique d'un entier de n bits,
- MEM_n: bloc mémoire pour stocker un entier de n bits.

Chapitre 8 : Attaques et Contremesures

C'est le cœur de la thèse. Ce chapitre liste les attaques physiques et contremesures sur ECC, avec une précision sur la description de chaque attaque et le coût de chaque contremesure en utilisant les notations du chapitre précédent.

Quand les designers proposent des méthodes pour se prémunir contre une classe d'attaques ou une attaque en particulier, les cryptanalystes proposent de nouvelles attaques pour contourner ou rendre totalement inefficace certaines contremesures. C'est réellement un jeu du chat et de la souris entre les attaques et les protections. Nous avons choisi de présenter les attaques et les contremesures avec une structure d'arbre pour correspondre à cette idée.

Nous décrivons ci-dessous nos attaques et contremesures nouvelles qui ont été publiées ou vont être publiées dans des conférences.

Same-Values Analysis sur la contremesure d'atomicité

La contremesure d'atomicité consiste à réécrire les formules d'addition et de doublement de point en utilisant les mêmes *patterns atomiques* [CCJ04]. Un pattern atomique est une séquence d'opérations dans le corps de base.

Cette contremesure contrecarre l'attaque par analyse de courant simple classique introduite dans [Cor99] car l'attaquant n'est pas capable de différencier une addition d'un doublement. Cette contremesure a été améliorée dans [GV10].

Notre proposition d'attaque est d'identifier les paires de multiplications où il y a un opérande en commun au sein de trois patterns atomiques seulement si ces trois patterns correspondent à une addition suivi d'un doublement. Cette suite d'opérations est effectuée si le bit courant du scalaire est différent de 0 lors de l'exécution de l'ECSM. Les opérandes en commun sont illustrés dans la Figure 2 avec des boîtes numérotées. Les opérandes en commun ont le même numéro. Seize paires de multiplications peuvent être analysées.

	ECADD - part 1	ECADD - part 2	modECDBL
1.	$T_1 \leftarrow \boxed{Z_2}^2_{1,2,14}$	$T_1 \leftarrow \boxed{T_6}^2_{9,10}$	$T_1 \leftarrow \boxed{X_1}^2_{12}$
2.	$\star$	$\star$	$T_2 \leftarrow Y_1 + Y_1$
3.	$T_2 \leftarrow Y_1 \times \boxed{Z_2}_{1,3,15}$	$T_4 \leftarrow T_5 \times T_1$	$Z_3 \leftarrow T_2 \times \boxed{Z_1}_{14,15,16}$
4.	$\star$	$\star$	$T_4 \leftarrow T_1 + T_1$
5.	$T_5 \leftarrow Y_2 \times \boxed{Z_1}_{4,5}$	$T_5 \leftarrow T_1 \times \boxed{T_6}_{9,11}$	$T_3 \leftarrow T_2 \times Y_1$
6.	$\star$	$\star$	$T_6 \leftarrow T_3 + T_3$
7.	$T_3 \leftarrow \boxed{T_1}_7 \times T_2$	$T_1 \leftarrow \boxed{Z_1}_{5,6} \times \boxed{T_6}_{10,11}$	$T_2 \leftarrow T_6 \times T_3$
8.	$\star$	$\star$	$T_1 \leftarrow T_4 + T_1$
9.	$\star$	$\star$	$T_1 \leftarrow T_1 + W_1$
10.	$T_4 \leftarrow \boxed{Z_1}^2_{4,6}$	$T_6 \leftarrow T_2^2$	$T_3 \leftarrow T_1^2$
11.	$T_5 \leftarrow T_5 \times \boxed{T_4}_8$	$Z_3 \leftarrow T_1 \times \boxed{Z_2}_{2,3,16}$	$T_4 \leftarrow T_6 \times \boxed{X_1}_{13}$
12.	$\star$	$T_1 \leftarrow T_4 + T_4$	$T_5 \leftarrow W_1 + W_1$
13.	$T_2 \leftarrow T_2 - T_3$	$T_6 \leftarrow T_6 - T_1$	$T_3 \leftarrow T_3 - T_4$
14.	$T_5 \leftarrow \boxed{T_1}_7 \times X_1$	$T_1 \leftarrow T_5 \times T_3$	$W_3 \leftarrow T_2 \times T_5$
15.	$\star$	$X_3 \leftarrow T_6 - T_5$	$X_3 \leftarrow T_3 - T_4$
16.	$\star$	$T_4 \leftarrow T_4 - X_3$	$T_6 \leftarrow T_4 - X_3$
17.	$T_6 \leftarrow \boxed{X_2}_{12,13} \times \boxed{T_4}_8$	$T_3 \leftarrow T_4 \times T_2$	$T_4 \leftarrow T_6 \times T_1$
18.	$T_6 \leftarrow T_6 - T_5$	$Y_3 \leftarrow T_3 - T_1$	$Y_2 \leftarrow T_4 - T_2$

Figure 2: Opérandes en commun dans seulement si les patterns correspondent à une addition de points (deux premières colonnes) suivi d'un doublement (troisième colonne)

Pour distinguer les opérandes en commun, nous proposons deux méthodes différentes. La première méthode fournit une attaque verticale, la seconde fournit une attaque horizontale plus puissante.

La première méthode consiste à analyser plusieurs traces correspondant à plusieurs ECSMs. Une paire de multiplication parmi les seize est choisie arbitrairement. Ensuite, deux variables

aléatoires sont construits à partir des traces. La variable X est construite à l'endroit temporelle correspondant à la première multiplication et la variable Y correspondant à la deuxième multiplication de la paire comme illustrée dans la Figure 3. Le coefficient de corrélation est calculé entre ces deux variables. Ce coefficient est élevé si pour chaque paire de multiplication, on a effectivement un opérande en commun.

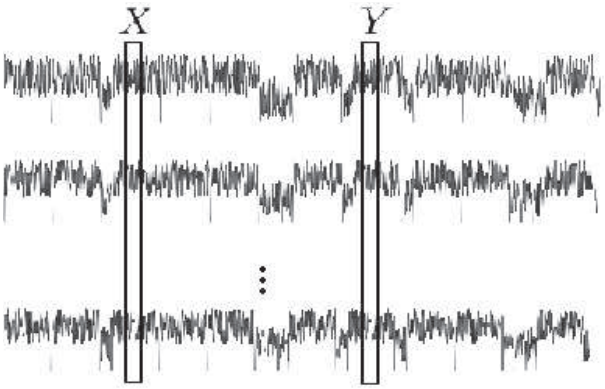


Figure 3: Construction des variables aléatoires pour une détection de même valeurs

Dans la deuxième méthode, nous utilisons une technique décrite dans l'attaque Big Mac [Wal01]. Soient T_1, T_2 les traces durant le calcul de respectivement deux multiplications $A \times B \bmod P$, $C \times D \bmod P$, avec $A \neq C$. À partir de T_1, T_2, la méthode Big Mac consiste à affirmer si $C = D$. Le succès de cette attaque dépend de la taille des entiers manipulés. Plus les entiers sont grands, plus les chances de succès de la distinction sont élevées. Ainsi, cette attaque fonctionne très bien sur RSA mais pas sur ECC car les entiers manipulés sont beaucoup plus petits (256 bits pour ECC face à 2048 pour RSA pour atteindre le même niveau de sécurité).

Nous avons étendu cette méthode pour attaquer la contremesure d'atomicité. En effet, comme indiqué précédemment, on peut comparer non pas une seule paire de multiplications mais seize. Le grand nombre de paires de multiplication que nous pouvons comparer permet de compenser la petite taille des entiers.

Nous avons testé cette attaque expérimentalement et nous avons obtenu de très bons résultats.

Décalage de la Courbe par Isomorphisme

Nous présentons notre contremesure contre l'attaque Refined Side-Channel Analysis (RSCA) [Gou03]. Cette attaque prend avantage de points particuliers de la forme $P_0 = (0, y)$. Ce point va apparaître durant le calcul de l'ECSM uniquement sous certaines conditions du scalaire secret.

Notre contremesure, publiée à [DGH+12], consiste à changer de courbe de départ par isomorphisme. L'isomorphisme φ est choisi de telle sorte que l'image du point de départ $P = (x_P, y_P)$ vaut $\varphi(P) = P' = (0, y_P)$.

Les courbes E and E' d'équations

$$\begin{aligned} E\colon &\quad y^2 = x^3 + a_4 x + a_6, \\ E'\colon &\quad y^2 = x^3 + a'_2 x^2 + a'_4 x + a'_6 \end{aligned}$$

sont isomorphiques sur $\mathbb{F}_p$ si et seulement s'ils existent $u \in \mathbb{F}_p^*$ et $r \in \mathbb{F}_p$ tels que le changement de variables

$$(x, y) \to (u^{-2}(x - r), u^{-3}y)$$

transforme équation E en l'équation E' avec :

$$\begin{cases} u^2 a'_2 &= 3r \\ u^4 a'_4 &= a_4 + 3r^2 \\ u^6 a'_6 &= a_6 + r a_4 + r^3 \ . \end{cases}$$

Si $P = (x_P, y_P)$ est le point de base, il suffit de choisir $u = 1$ et $r = x_P$ pour obtenir le résultat attendu. Cet isomorphisme est illustré à la Figure 4.

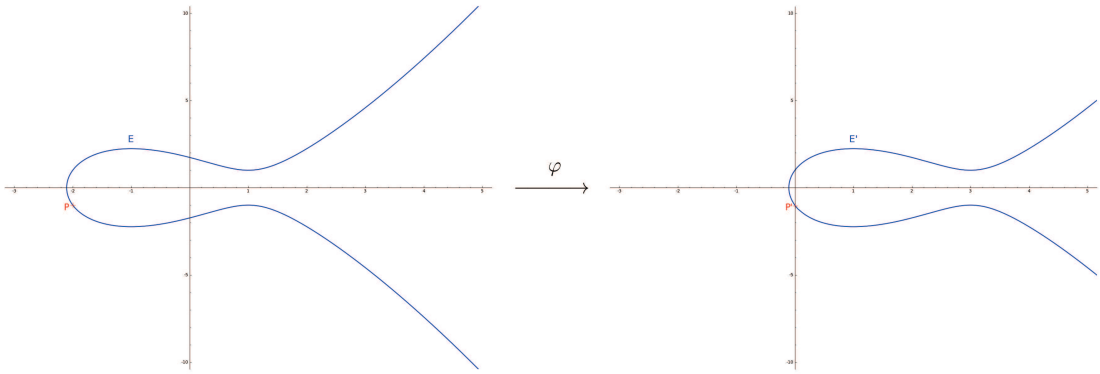


Figure 4: Décalage par isomorphisme avec E: $y^2 = x^3 - 3x + 3$ et E': $y^2 = x^3 - 6x^2 + 9x + 1$.

Malheureusement, E' n'est pas sous sa forme d'équation de Weierstraß réduite. Les formules classiques ne s'appliquent pas. Il faut les modifier et elles ont un surcoût. Cependant, ce surcoût est compensé par le fait que le point de base est $P' = (0, y_P)$. Nous pouvons simplifier les formules lorsqu'on additionne un point avec P' ou $-P'$. Pour certains ECSMs, il est même avantageux d'appliquer la contremesure.

La sécurité contre l'attaque RSCA est assurée par des théorèmes que nous détaillons dans le mémoire. L'idée est que P' et $-P'$ sont les seules points ayant une coordonnée x à zéro. Le point P' est le point de base de l'ECSM, ainsi ce point ne peut apparaître en tant que point intermédiaire en faveur de l'attaquant.

Same-Values Analysis classique

Nous décrivons notre attaque publiée à [MGD+12]. L'idée de l'attaque est de profiter de points spéciaux. Ces points ont la particularité suivante. Pendant leur doublement, deux variables intermédiaires ont la même valeur.

L'attaquant identifie un point particulier P_{SVA}. Il choisit le point de base de l'ECSM de telle sorte P_{SVA} apparaisse pendant l'ECSM seulement si le bit courant visé est égal à 1. L'attaquant déduit ainsi le scalaire secret de façon récursive.

Pour détecter si le point particulier est effectivement apparu durant le calcul de l'ECSM, nous proposons deux méthodes différentes.

La première correspond à une attaque verticale car plusieurs traces sont nécessaires. Cette méthode est similaire à l'attaque verticale contre la contremesure de l'atomicité et est illustré à la Figure 3.

Pour la seconde méthode, il nous faut décrire des points ayant des conditions plus fortes que précédemment. En plus de faire intervenir deux mêmes valeurs durant le doublement, ces valeurs sont ensuite utilisées de la même façon. Typiquement, ces valeurs seront mises au carré. La différence des traces correspondant au calcul du carré modulaires des deux valeurs permet de révéler si les valeurs sont justement identiques.

Attaque sur la conversion de coordonnées projectives en coordonnées affines

Cette attaque a été présentée à [MMNT13]. Les coordonnées projectives ou Jacobiennes permettent d'accélérer les calculs durant l'ECSM. À la fin du calcul, le résultat est converti en coordonnées affines. Notre attaque consiste à injecter une faute pendant la conversion et récupérer les coordonnées projectives ou Jacobiennes du point $Q = [k]P$. Naccache, Smart et Stern ont montré que lorsque le résultat de l'ECSM était fourni en coordonnées projectives ou Jacobiennes, l'attaquant peut retrouver quelques bits de k [NSS04].

La procédure suivante convertit le point en coordonnées Jacobiennes $P = (X, Y, Z) = (xZ^2, yZ^3, Z)$ en coordonnées affines (x, y).

$$\text{CONVERT}(X,Y,Z) = \begin{cases} r \leftarrow Z^{-1} \\ s \leftarrow r^2 \\ x \leftarrow X \cdot s \\ t \leftarrow Y \cdot s \\ y \leftarrow t \cdot r \quad \text{return}(x,y) \end{cases}$$

Notre attaque consiste à injecter une faute juste après l'étape $s \leftarrow r^2$. La valeur corrompue $s + \epsilon$ donne les équations suivantes :

$$\tilde{x} = X(s+\epsilon) \Rightarrow \tilde{x} = x + xZ^2\epsilon,$$
$$\tilde{y} = Y(s+\epsilon)r \Rightarrow \tilde{y} = y + yZ^2\epsilon \ .$$

Ensuite, nous proposons différentes méthodes pour récupérer la valeur de Z manquante à partir de un ou plusieurs résultats incorrects $\tilde{x}, \tilde{y}$. Les méthodes proposées dépendent de la taille de la faute.

Notre attaque permet de récupérer les coordonnées projectives ou Jacobiennes du résultat de l'ECSM et ainsi appliquer l'attaque de [NSS04].

Chapitre 9: Attaques en Faute Différentielles sur ECDSA

Certaines attaques en faute différentielles nécessitent de comparer un résultat corrompu avec le résultat correct d'un ECSM pour déduire quelques bits du scalaire.

Lors d'une signature ECDSA (Algorithme 1), le scalaire est choisi aléatoirement pour chaque nouvelle signature. Ainsi, la comparaison entre un résultat de ECSM corrompu et le bon résultat semble infaisable.

Nous avons remarqué que c'est en fait possible. Grâce à un résultat de ECSM corrompu, fourni avec la signature corrompue, il est possible de récupérer le bon résultat de l'ECSM. Voici la méthode utilisée.

Nous supposons que, durant la procédure de signature, une faute est introduite pendant le calcul de $Q = [k]G = (x_Q, y_Q)$, ce qui donne le mauvais résultat $\tilde{Q} = (x_{\tilde{Q}}, y_{\tilde{Q}})$. La signature incorrecte $(\tilde{r}, \tilde{s})$ vérifie:

$$\tilde{r} = x_{\tilde{Q}} \bmod t$$
$$\tilde{s} = k_{inv}(d\tilde{r} + m) \bmod t$$

À partir de $(\tilde{r}, \tilde{s})$, l'attaquant peut calculer :

$$\begin{aligned}
\tilde{w} &= \tilde{s}^{-1} \bmod t \\
\tilde{u}_1 &= \tilde{w} \cdot m \bmod t \\
\tilde{u}_2 &= \tilde{w} \cdot \tilde{r} \bmod t \\
R &= [\tilde{u}_1]G + [\tilde{u}_2]P = \left[\tfrac{km}{d\tilde{r}+m}\right] G + \left[\tfrac{k\tilde{r}}{d\tilde{r}+m}\right] P = \left[\tfrac{km}{d\tilde{r}+m}\right] G + \left[\tfrac{dk\tilde{r}}{d\tilde{r}+m}\right] G \\
&= \left[k \cdot \tfrac{d\tilde{r}+m}{d\tilde{r}+m}\right] G = [k]G
\end{aligned}$$

Le point R est en fait la bonne valeur de $Q = (x_Q, y_Q) = [k]G$ qui était censée être calculée au départ. L'attaquant retrouve x_Q, y_Q et $x_{\tilde{Q}} \bmod t$. Il peut ainsi appliquer certains attaques en faute différentielles car il peut comparer le bon et le mauvais résultat.

Chapitre 10: Résumé des Attaques de Façon Contextuel

Dans le chapitre précédent, pour chaque attaque décrite, nous fournissons le contexte de l'attaque. Dans ce chapitre-ci, une synthèse des attaques au niveau contextuelle est donnée. Cette synthèse est très utile pour un designer car il peut vérifier très rapidement si des attaques sont faisables suivant l'application qu'il est censé développer. Si certaines attaques ne sont pas faisables, une protection n'est pas nécessaire et l'application peut ainsi être plus rapide.

Chapitre 11: Synthèse des Attaques et des Contremesures

Ce chapitre donne une synthèse de l'efficacité de chaque contremesure contre chaque attaque. Toutes les attaques et contremesures de cette thèse sont répertoriées sous forme d'un tableau pour un affichage clair de l'interaction entre les attaques et les contremesures.

Conclusion et Perspectives

Dans cette thèse, nous avons étudié les attaques physiques sur ECC. Pour chaque attaque, nous détaillons le contexte dans lequel l'attaque est faisable. Aussi, pour chaque contremesure, nous détaillons son coût. Nous avons choisi de présenter les attaques et les contremesures suivant une structure d'arborescence pour clairement indiquer si une attaque a été présentée contre une contremesure en particulier ou si elle est plus générale. De la même façon, nous pouvons voir si une contremesure couvre une attaque en particulier ou si elle est plus générale.

Nous introduisons de nouvelles attaques appelées Same-Values Analysis. Les attaques sont nommées ainsi car elles se basent sur des mêmes valeurs qui sont répétées au sein d'un ECSM. Elles différent sur l'implémentation visée ou sur la méthode de détection de l'apparition des mêmes valeurs.

Nous introduisons également une nouvelle attaque en faute différentielle. À la différence des autres attaques en faute, où la faute est introduite avant ou pendant l'ECSM, une faute est ici introduite pendant la conversion de coordonnées projectives à affines à la fin du calcul.

Nous présentons également une contremesure contre l'attaque Refined Side-Channel Analysis (RSCA). L'attaque RSCA repose sur l'apparition d'un point particulier de la forme $(0, y)$. Nous proposons d'utiliser un isomorphisme entre les courbes elliptiques pour contrôler le point particulier gênant. Son apparition ne révèle rien sur le scalaire secret.

Enfin, nous montrons que les attaques en faute différentielles peuvent s'appliquer sur ECDSA, qui est un protocole de signature probabiliste.

Dans cette thèse, nous fournissons un état-de-l'art complet sur les attaques physiques sur ECC. À l'avenir, de nouvelles attaques vont inévitablement survenir. De même, de nouvelles contremesures émergeront. Elles peuvent être intégrées au fur et à mesure dans l'état-de-l'art en suivant la même méthodologie.

Il serait intéressant d'étendre ce travail à d'autres cryptosystèmes asymétriques tels que RSA ou ceux basés sur le couplage. Une autre idée serait de présenter un état-de-l'art commun entre ces cryptosystèmes d'une certaine façon. En effet, de nombreuses similarités subsistent entre ces cryptosystèmes. D'abord, le même module d'arithmétique modulaire est généralement utilisé pour RSA, ECC et la cryptographie à base de couplage. Ensuite, les méthodes de multiplication scalaire pour ECC et le couplage sont similaires aux méthodes d'exponentiations pour RSA. Ainsi, des attaques et contremesures sont également similaires.

Table of Contents

List of Acronyms

AES	Advanced Encryption Standard
CSCA	Correlation Side-Channel Analysis
CRC	Cyclic Redundancy Check
CRT	Chinese Remainder Theorem
DES	Data Encryption Standard
DFA	Differential Fault Attack
DSCA	Differential Side-Channel Analysis
ECADD	Elliptic Curve Addition
ECC	Elliptic Curve Cryptography
ECDBL	Elliptic Curve Doubling
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECSM	Elliptic Curve Scalar Multiplication
NAF	Non-Adjacent Form
RNG	Random Number Generation
RPG	Random Permutation Generation

RSCA Refined Side-Channel Analysis

SCA Side-Channel Analysis

SSCA Simple Side-Channel Analysis

SVA Same-Values Analysis

ZADDC Conjugate co-Z Addition

ZADDC' (X, Y)-only Conjugate co-Z Addition

ZADDU co-Z Addition and Update

ZADDU' (X, Y)-only co-Z Addition and Update

ZSCA Zero Side-Channel Analysis

Introduction

Cryptography formerly referred to the art of protecting messages. Before sending a message, it is first encoded in a way that makes it seem nonsensical. This process is called *encryption*. Only the intended receiver can read it by applying the inverse process, the *decryption*. The processes are known by the sender and the receiver only. Confidentiality of the message is then ensured.

Following the evolution of telecommunications, cryptography also changed. Modern cryptography, highly influenced by the *Data Encryption Standard* (DES) in the 1970s, is no more an art but a science. Today, in addition to confidentiality, cryptography brings new functionalities such as *authentication*, *i.e.* the confirmation of an entity's identity, and *integrity*, *i.e.* the confirmation that messages received have not been modified by an unauthorized entity, and many others.

By Kerckhoffs's principle, the security of a cryptosystem should rely on secret data only, rather than on the very methods used for encoding and decoding messages. These methods will necessarily be discovered somehow. Such a secret is called a *key*. Two families of cryptosystems arise: *symmetric* and *asymmetric cryptosystems*.

In a symmetric cryptosystem, the same key is used for both encryption and decryption. It must be kept secret by the entities who want to communicate. Nowadays, encryption and decryption are very fast and are suitable for confidentiality. The DES is in fact a symmetric cryptosystem.

In an asymmetric cryptosystem, two keys are used. One key is broadcast. It is used for public processes such as encryption or verification of signatures. This key is called the *public key*. When a message is encrypted, no one can decrypt it, except the owner of the second key called the *private key*. The owner of the private key is also the only one who can sign messages. Asymmetric cryptosystems are interesting because they are suitable for authentication. However, the computations they involve are a lot slower than those involved by symmetric cryptosystems. This is because they rely on computationally costly mathematical tools such as arithmetic on very large integers. RSA is the first asymmetric cryptosystem, introduced by Rivest, Shamir and Adleman in 1977. Koblitz and Miller independently introduced the use of elliptic curves for cryptographic applications in the late 1980s. The class of asymmetric cryptosystems based on elliptic curves is called Elliptic Curve Cryptography (ECC). This thesis focuses on ECC only.

Nowadays, the security of a cryptosystem is generally based on the difficulty of solving a mathematical problem, such as integer factorization or the Elliptic Discrete Logarithm Problem in the case of ECC. Proofs based on assertions such as "this cryptosystem C is unbreakable if no one can solve problem P" arise. These proofs go with a cryptosystem and ensure its security.

Cryptography is naturally used in smart cards. Many different examples such as bank cards, telephony, or electronic passports, obviously need cryptography to ensure confidentiality, authentication or integrity. A new kind of attacks in smart cards arose in the late 1990s. Kocher showed that analysing the execution timing of a device to execute a cryptographic algorithm can be enough to recover the secret key. No need to solve the difficult underlying mathematical problem. Other attacks of this kind have emerged since then. They are based on the observation of the device's behaviour when a cryptographic algorithm is being executed. This kind of attacks is called *physical attacks*. The security proofs of a cryptosystem are no longer enough when physical attacks are taken into account. Of course, many methods have also emerged to prevent these attacks.

Provable security against physical attacks is still a research topic. Until a clear methodology of provable security on ECC in smart cards is accepted by the cryptographic community, attacks and countermeasures will be a cat-and-mouse game. So, to date, a designer who wants to prove the security of his implementation, has no choice but to argue that his implementation is protected against *all* existing physical attacks by an exhaustive method.

This thesis is intended to help the designer implementing a secure embedded ECC. The first part gives a detailed background on ECC, useful to understand the physical attacks and countermeasures. In the second part, the physical attacks and countermeasures are displayed with a tree structure. This follows the idea of the cat-and-mouse game of attacks and countermeasures. In addition to the complete and accurate state-of-the art, we introduce new attacks. Some of these attacks are powerful since they need only one execution of the cryptographic algorithm. Moreover the knowledge of the inputs is not necessary. Also, a new countermeasure against an attack, the Refined Power Analysis is proposed. Surprisingly, under some assumptions that we detail, it turns out that applying the countermeasure is in fact more efficient. In addition to security, this is an improvement for ECC implementations.

Part I

Elliptic Curve Cryptography

Introduction

The use of elliptic curves for cryptographic applications has been independently introduced by Koblitz [Kob87] and Miller [Mil85]. With the same security level, Elliptic Curve Cryptography (ECC) involves smaller key lengths compared with other asymmetric cryptosystems such as RSA or systems based on the multiplicative group of a finite field. In most architectures, the complexity in time of the modular multiplication, which is the most frequently used operation in both ECC and RSA, is quadratic in the size of the operands. For example, for a security requirement of 128 bits, the minimal size of the key in ECC is 256 bits, as opposed to RSA in which the public modulus is at least a 3072-bit integer. For this reason, the use of elliptic curves in cryptographic applications has increased these last years, especially in embedded systems with limited resources.

The elliptic curves used for ECC are generally defined over prime fields $\mathbb{F}_p$, $p > 3$ or binary fields $\mathbb{F}_{2^n}$. The latter can bring better performance because arithmetic operations in such fields are carry-free. However, elliptic curves over prime fields are more often used because of the following main reasons. First, elliptic curves over binary fields are restricted by several patents. The second reason is that, recently, Faugères, Perret, Petit and Renault improved the index calculus to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP) on elliptic curves over binary fields [FPPR12]. They did not break ECC over binary fields but it is a strong basis towards future results to reduce the complexity of the ECDLP. Consequently, we only focus on ECC over $\mathbb{F}_p$, $p > 3$.

This part exposes the background on ECC. Chapter 1 gives the general definition of elliptic curves. The different levels of ECC's hierarchy are described in the next chapters. The arithmetic in $\mathbb{F}_p$ is described in Chapter 2. The most efficient elliptic curve formulæ can be found in Chapter 3. The most frequently used elliptic curve scalar multiplications are given in Chapter 4. Some examples of cryptographic protocols, including a signature, a cipher and a key agreement schemes, based on elliptic curves can be found in Chapter 5. Finally, the security of ECC in the black box model is discussed in Chapter 6.

Chapter 1

General Definition

This chapter gives the background on elliptic curves used for cryptographic applications. The following definitions and properties can be found in [BSS99, CFA+06].

1.1 Elliptic Curves in Affine Coordinates

Definition 1.1. In a field $\mathbb{K}$, an elliptic curve is defined by its Weierstraß equation:

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ . \tag{1.1}$$

with $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ and $\Delta \neq 0$ where Δ is defined as follows:

$$\left\{\begin{array}{lcl} d_2 & = & a_1^2 + 4a_2 \\ d_4 & = & 2a_4 + a_1a_3 \\ d_6 & = & a_3^2 + 4a_6 \\ d_8 & = & a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\ \Delta & = & -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \ . \end{array}\right.$$

We denote by $E(\mathbb{K})$ the set of points $(x, y) \in \mathbb{K}^2$ satisfying Equation (1.1), together with a "point at infinity" $\mathcal{O}$. Δ is called the discriminant of the curve. The coordinates x, y are called the *affine* coordinates.

Remark 1.2. If the field is implicit, the set of points is denoted by E.

1.2 Group Structure

$E(\mathbb{K})$ is an additive Abelian group defined by the following addition law. Let $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \notin \{\mathcal{O}, -P\}$ be two points on $E(\mathbb{K})$. The point $R = (x_3, y_3) = P + Q$ is defined by the formula:

$$\begin{array}{l} x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3 \end{array} \quad \text{where } \lambda = \left\{\begin{array}{ll} \frac{y_1 - y_2}{x_1 - x_2} & \text{if } P \neq Q, \\ \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{if } P = Q. \end{array}\right.$$

The inverse element of the point P is $-P = (x_1, -y_1 - a_1x_1 - a_3)$. $\mathcal{O}$ is the neutral element, in that $P + \mathcal{O} = \mathcal{O} + P = P$.

Geometrically, we draw the line passing through P, Q (or the tangent of P if $P = Q$). λ is the slope of this line. The line intersects the curve in a third point, counting with multiplicity. We call it $S = (x_3, y_4)$. x_3, y_4 are found by solving the equations system

$$\begin{cases} y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \\ y = \lambda x + y_1 - \lambda x_1 \end{cases}$$

This leads to the equation in x:

$$(\lambda x + y_1 - \lambda x_1)^2 + a_1x(\lambda x + y_1 - \lambda x_1) + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ .$$

We can simplify the equation since x_1, x_2 are solutions. This gives the formula of x_3 above. x_3 yields y_4. We take $-S = R = (x_3, y_3 = -y_4 - a_1x_3 - a_3)$ as the result of the addition. R is the intersection between the curve and the line passing through $\mathcal{O}$ and S.

Remark 1.3. We take $R = -S$ as the result of the addition to respect the group axioms.

If $\mathbb{K}$ is a finite field of q elements, the number of points on $E(\mathbb{K})$ is denoted by $\#E(\mathbb{K})$ (or simply $\#E$ if the field is implicit). By Hasse's theorem [Has36], $\#E(\mathbb{K})$ satisfies

$$|\#E(\mathbb{K}) - q - 1| \leq 2\sqrt{q} \ .$$

$\#E(\mathbb{K})$ and q have the same magnitude. In cryptographic applications, recommended elliptic curves satisfy $\#E(\mathbb{K}) = ht$ with t a large prime and h a very small number (1, 2 or 4). h is called the *cofactor*. Only points of order t are considered in cryptographic applications.

1.3 Short Weierstraß Equation

Five parameters define an elliptic curve in its Weiertraß equation. The group law requires many field operations. Using isomorphisms between elliptic curves simplify the curve equation and thus the group law.

Definition 1.4. Two elliptic curves E, E' defined by their Weierstraß equations:

$$\begin{array}{ll} E: & y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \\ E': & y^2 + a'_1xy + a'_3y = x^3 + a'_2x^2 + a'_4x + a'_6, \end{array}$$

are said to be *isomorphic* over $\mathbb{K}$ if there exist $r, s, t \in \mathbb{K}, u \in \mathbb{K}^*$, such that the change of variables

$$x' = u^{-2}(x - r), y' = u^{-3}(y - sx - sr - t) \tag{1.2}$$

preserving $\mathcal{O}$, transforms equation E into equation E'. The transformation (1.2) is called an *admissible change of variables*. Furthermore, the elliptic curves parameters are linked by

$$\begin{cases} ua'_1 &= a_1 + 2s \\ u^2a'_2 &= a_2 - sa_1 + 3r - s^2 \\ u^3a'_3 &= a_3 + ra_1 + 2t \\ u^4a'_4 &= a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st \\ u^6a'_6 &= a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1 \ . \end{cases}$$

Let the curve $E\colon y^2+a_1xy+a_3y = x^3+a_2x^2+a_4x+a_6$ defined over a field $\mathbb{K}$ of characteristic different from 2 and 3 (which makes division by 2 and 3 possible). Choosing

$$\begin{cases} u &= 1 \\ r &= -\frac{a_1^2+4a_2}{12} \\ s &= -\frac{a_1}{2} \\ t &= \frac{a_1^3+4a_1a_2-12a_3}{24} \end{cases}$$

transforms the curve E into

$$E'\colon y^2 = x^3 + ax + b \ . \tag{1.3}$$

for some $a, b \in \mathbb{K}$. Equation (1.3) is called the short Weierstraß equation. Its discriminant is $\Delta = -16(4a^3 + 27b^2)$.

Let $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \notin \{\mathcal{O}, -P\}$ be two points on $E'(\mathbb{K})$. Points addition $R = (x_3, y_3) = P + Q$ is given by the formula:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \quad \text{where } \lambda = \begin{cases} \frac{y_1-y_2}{x_1-x_2} & \text{if } P \neq Q, \\ \frac{3x_1^2+a}{2y_1} & \text{if } P = Q. \end{cases}$$

The inverse element of P is $-P = (x_1, -y_1)$. The group law of elliptic curves in the short Weierstraß equation is illustrated in Figure 1.1 over the real elements.

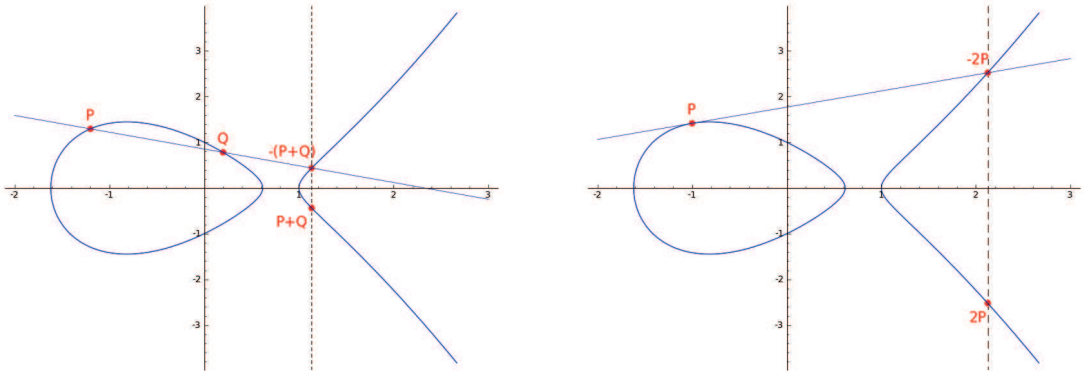


Figure 1.1: Addition and doubling of points on the curve $y^2 = x^3 - 2x + 1$ over $\mathbb{R}$

1.4 Elliptic Curves in Projective Coordinates

To avoid costly inversions when computing λ, elliptic curves can also be defined in the projective plane.

Definition 1.5. The projective plan $\mathbb{P}^2(\mathbb{K})$ over the field $\mathbb{K}$ is:

$$\mathbb{P}^2(\mathbb{K}) = \frac{\{(X, Y, Z) \in \mathbb{K}^3 \setminus (0, 0, 0)\}}{\sim}$$

with $\sim$ being the following equivalence relation:

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \text{ if } \exists\ r \in \mathbb{K}^* \text{ such that } (X_1, Y_1, Z_1) = (rX_2, rY_2, rZ_2)$$

The coordinates are called *homogeneous* or *projective coordinates*. The equivalences classes, denoted by $(X : Y : Z)$ with $(X, Y, Z) \in \mathbb{K}^3 \setminus (0, 0, 0)$, are called *projective points*. We denote by (X, Y, Z) a *representative* of the projective point $(X : Y : Z)$.

If $\mathbb{K}$ is finite of cardinality $\#\mathbb{K}$, the projective point $(X : Y : Z)$ have $\#\mathbb{K}-1$ representatives. If $Z \neq 0$, $(X : Y : Z)$ corresponds to the unique affine point $(X/Z, Y/Z)$.

Remark 1.6. In the rest of this thesis, by abuse of notation, we will call a "point" on the curve which is in fact a "representative" of a projective point. Unless otherwise specified (explicitly stated or using the notation $(X : Y : Z)$), a point will refer to a representative of a projective point.

The equation of an elliptic curve in the homogeneous coordinates system in the reduced Weierstraß form is:

$$E^{\mathcal{P}} : Y^2Z = X^3 + aXZ^2 + bZ^3 \ .$$

The point at infinity $\mathcal{O}$, which is not explicit in affine coordinates, is the projective point $\mathcal{O} = (0 : 1 : 0)$. In fact, $\mathcal{O}$ is the only projective point on the curve with a zero Z coordinate.

The coordinates were generalized using different equivalence relations. The Jacobian coordinates is often used for efficiency reason. The equation of an elliptic curve in the Jacobian projective coordinates system in the reduced Weierstraß form is:

$$E^{\mathcal{J}} : Y^2 = X^3 + aXZ^4 + bZ^6 \ .$$

The point at infinity is $\mathcal{O} = (1 : 1 : 0)$. The projective point $(X : Y : Z)$, with $Z \neq 0$, corresponds to the unique affine point $(X/Z^2, Y/Z^3)$. The equivalence relation is:

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \text{ if } \exists\ r \in \mathbb{K}^* \text{ such that } (X_1, Y_1, Z_1) = (r^2X_2, r^3Y_2, rZ_2)$$

Remark 1.7. We omit the notation $\mathcal{J}$ or $\mathcal{P}$ when the coordinates system is obvious in the context.

Let $P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2)$ be two points on $E^{\mathcal{J}}(\mathbb{K})$ with $P_1 \neq \mathcal{O}, ord(P_1) > 2$ and $P_2 \notin \{\mathcal{O}, -P_1\}$. Point doubling and points addition are defined by the following formulæ:

- **ECDBL**. $P_3 = (X_3, Y_3, Z_3) = 2P_1$ can be computed as:
  $X_3 = T,\ Y_3 = -8Y_1^4 + M(S - T),\ Z_3 = 2Y_1Z_1,$
  $S = 4X_1Y_1^2,\ M = 3X_1^2 + aZ_1^4,\ T = -2S + M^2$
- **ECADD**. $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$ can be computed as:
  $X_3 = -H^3 - 2U_1H^2 + R^2,\ Y_3 = -S_1H^3 + R(U_1H^2 - X_3),\ Z_3 = Z_1Z_2H,$
  $U_1 = X_1Z_2^2,\ U_2 = X_2Z_1^2,\ S_1 = Y_1Z_2^3,\ S_2 = Y_2Z_1^3,\ H = U_2 - U_1,\ R = S_2 - S_1$

There is a certain hierarchy of calculation in ECC applications. The different levels, from the lowest to the highest, are finite field arithmetic, elliptic curve arithmetic, elliptic curve scalar multiplication (ECSM) and cryptographic protocol. These different levels are detailed in the next chapters.

Chapter 2

Finite Field Arithmetic

Finite field arithmetic finds its application in asymmetric cryptography. Therefore, it is a subject under intensive study, and many different algorithms and implementations are proposed to perform fast operations in the field $\mathbb{F}_p$. A description of certain methods is required to understand some side-channel attacks targeting this level in ECC's hierarchy. This chapter details these methods.

2.1 Field Element Representation

The elements of $\mathbb{F}_p$ are integers in $[0, p-1]$. The algorithms to perform field operations generally operate word by word. Denote w the size of the words[1].

Denote $n = \lceil \log_2 p \rceil$ the bit length of p and $m = \lceil n/w \rceil$. The integers in $[0, p-1]$ are manipulated as arrays of m elements, as illustrated in Figure 2.1.

$a[m-1]$	$\ldots$	$a[1]$	$a[0]$

Figure 2.1: Representation of $A = 2^{(m-1)w}a[m-1] + \cdots + 2^w a[1] + a[0]$ as an array of m words.

In the rest of this chapter, capital letters will denote long integers and lower cases will denote words of w bits.

2.2 Main Module

In the algorithms exposed in the rest of this chapter, we suppose there is a module performing the following operation:

$$(u, v) \leftarrow a \times b + e + c \tag{2.1}$$

with e, a, b, c, u, v being w-bit words. v is the low word of the result and u is the high word (called the carry). No overflow occurs because $a, b, c, e \in [0, 2^w[\Rightarrow a \times b + e + c \leq (2^w - 1)^2 + 2 \times 2^w - 2 = 2^{2w} - 1 \Rightarrow u < 2^w$. Thus, $a \times b + e + c = 2^w u + v$. We use the notation $(u, \cdot) \leftarrow a \times b + e + c$ when only the carry is updated; the low result is thrown out. In most architectures, the same module supports the two's complement of integers:

$$(u, v) \leftarrow a \times b + \bar{e} + c \tag{2.2}$$

[1] $w = 32$ or $w = 64$ in common architectures, even in smart cards where there are hardware accelerators.

The two's complement of e is $\bar{e} = 2^w - 1 - e$. Subtracting two words a, e with $a \geq e$ can be done using the module illustrated in Equation (2.2) by replacing b by 1 and c by 1. v will contain $a - e$ and u will be equal to 1 which is ignored.

2.3 Modular Addition and Subtraction

We give the schoolbook algorithms for addition and subtraction of long integers.

Algorithm 2 Schoolbook long integer addition (ADD)

Input: $A = (a[m-1], \ldots, a[0]), B = (b[m-1], \ldots, b[0]), m$.
Output: (c, R) with $R = A + B$ if $c = 0$ and $R = A + B - 2^{wm}$ if $c = 1$

1: $c \leftarrow 0$
2: **for** $i = 0$ **to** $m - 1$ **do**
3: $\quad (c, r[i]) \leftarrow a[i] + b[i] + c$ ▷ $c \in \{0, 1\}$ because $a[i] + b[i] + c \leq 2^{w+1} - 1$
4: **return** (c, R)

The operation of line 3 can be done using the module illustrated in Equation (2.1) by replacing b by 1 and e by b.

Algorithm 3 Two's complement schoolbook long integer subtraction (SUB)

Input: $A = (a[m-1], \ldots, a[0]), B = (b[m-1], \ldots, b[0]), m$.
Output: (c, R) with $R = A - B$ if $c = 0$ and $R = A - B + 2^{wm}$ if $c = 1$

1: $c \leftarrow 1$ ▷ e is set to 1 because we will compute $(A + \bar{B} + 1) = A - B + 2^{mw}$
2: **for** $i = 0$ **to** $m - 1$ **do**
3: $\quad (c, r[i]) \leftarrow a[i] + \bar{b[i]} + c$ ▷ $c \in \{0, 1\}$
4: $c \leftarrow c \oplus 1$
5: **return** (c, R)

With those two basic procedures, we can give modular addition and subtraction.

Algorithm 4 Modular Addition

Input: A, B, P, m such that $A < P, B < P$.
Output: $A + B \bmod P$

```
(c, R) ← ADD(A, B, m)
if c = 1 then
    (d, S) ← SUB(R, P, m)
    return S
else
    if R ≥ P then
        (d, S) ← SUB(R, P, m)
        return S
    else
        return R
```

Algorithm 5 Modular Subtraction

Input: A, B, P, m such that $A < P, B < P$.
Output: $A - B \bmod P$
$(c, R) \leftarrow \text{SUB}(A, B, m)$
if $c = 1$ **then**
$(d, S) \leftarrow \text{ADD}(R, P, m)$ ▷ in this case, $c = d = 1$ and the carry cancels the borrow
return S
else
return R

The conditional subtraction or addition of the previous algorithms can be inconvenient. First, an operator comparing long integers is required for the modular addition. Secondly, we will see in Sections 8.1 and 8.2.2.1 that some physical attacks take advantage of this step. We can remove the conditional step by introducing a dummy operation, as described below. We present the following methods to properly handle the carry and the borrow.

Algorithm 6 Constant Time Modular Addition (MADD)

Input: A, B, P, m such that $A < P, B < P$.
Output: $A + B \bmod P$
1: $(c, R) \leftarrow \text{ADD}(A, B, m)$
2: $(d, S) \leftarrow \text{SUB}(R, P, m)$
3: **if** $c \oplus d = 0$ **then**
4: **return** S
5: **else**
6: **return** R

The correctness of Algorithm 6 is proved below.
Since $0 \leq A < P, 0 \leq B < P$, we must consider the three following cases:

- $A + B < P < 2^{wm}$.
  In this case, $R = A + B$.
  Since $A + B < P$, $\implies c = 0$ and $d = 1$. R is therefore the correct result because it satisfies $0 \leq R = A + B < P$.

- $P < A + B < 2^{wn}$.
  In this case, $R = A + B$ and $S = A + B - P$.
  Since $P < A + B < 2^{wn} \implies c = 0$ and $d = 0$. Moreover, $P \leq A + B < 2P \implies 0 \leq A + B - P < P$. S is the only integer in $[0, P[$ satisfying $S = A + B \bmod P$.

- $2^{wn} < A + B < 2P$.
  In this case, $R = A + B - 2^{wn}$.
  Since $2^{wn} < A + B \implies c = 0$. Moreover, $A + B < 2P \implies A + B - 2^{wn} < 2P - 2^{wn} < P$. Therefore, $d = 1$ and $S = R - P + 2^{wn} = A + B - P$. S is the only integer in $[0, P[$ satisfying $S = A + B \bmod P$.

Remark 2.1. The condition $c = 1, d = 0$, never occurs.

Algorithm 7 Constant Time Modular Subtraction (mSUB)

Input: A, B, P, m such that $A < P, B < P$.
Output: $A - B \bmod P$
  $(c, R) \leftarrow \text{SUB}(A, B, m)$
  $(d, S) \leftarrow \text{ADD}(R, P, m)$
  if $c = 0$ **then**
    return R
  else ▷ in this case, $c = d = 1$ and the carry cancels the borrow
    return S

The correctness of Algorithm 7 is trivial considering Algorithm 5.

2.4 Modular Multiplication

Field inversions are costly compared with the other field operations. In Section 1.4, we explained how the Projective or Jacobian coordinates enables the substitution of inversions for multiplications. Hence, modular multiplications become the most numerous expensive operations involved in ECC. Great care should be taken for implementing the modular multiplication operator.

Many methods exist to efficiently perform modular multiplications. For instance:

- the Montgomery multiplication [Mon85],
- the Barrett multiplication [Bar86],
- the Quisquater's multiplication [Qui90, Qui91] and
- the Montgomery multiplication in the Residue Number System [Baj98].

In addition, some standardized elliptic curves, such as the ones proposed in the *Digital Signature Standard* [FIPS186-3], are based over prime fields enabling fast modular reduction. Due to the particular form of the prime p, modular reductions can be performed with only shifts, additions and subtractions. However, this optimization is not always considered because of the lack of genericity: each curve has its own modular multiplication module. The need of a generic modular multiplication module is required anyway for some ECC protocols. Moreover, a common module is generally implemented for all asymmetric cryptosystems.

We choose to detail the Montgomery multiplication (Algorithm 8), because it is quite common and many side-channel attacks are described with this example.

The algorithm, called the Coarsely Integrated Operand Scanning, alternates the multiplication and the reduction loops. This permits to avoid the extra memory required if a naive modular multiplication is performed: first compute the entire product, then reduce. This algorithm is from [KA96], where we added the final conditional subtraction step.

Algorithm 8 Montgomery Modular Multiplication (montMUL)

Input: $A = (a[m-1], \ldots, a[0]), B = (b[m-1], \ldots, b[0]), P = (p[m-1], \ldots, p[0]), \tilde{p}, m$ such that $A < P, B < P, \tilde{p} = -p[0] \bmod 2^w$.
Output: $A \times B \times 2^{-wm} \bmod P$

$R \leftarrow 0$
$s \leftarrow 0$ ▷ s will hold the $(m+1)^{th}$ word of the temporary result R

▷ **Main loop**
for $i = 0$ **to** $m-1$ **do**
▷ Step $R \leftarrow R + a[i]B$
  $c \leftarrow 0$
  for $j = 0$ **to** $m-1$ **do**
    $(c, r[j]) \leftarrow a[i] \times b[j] + r[j] + c$
  end for
  $(c, s) \leftarrow s + c$
  $t \leftarrow c$ ▷ t will hold the $(m+2)^{th}$ word of the temporary result R
▷ End Step $R \leftarrow R + a[i]B$
  $c \leftarrow 0$
  $(c, q) \leftarrow r[0] \times \tilde{p} + c$ ▷ $q = -RP^{-1} \bmod 2^w$

▷ Step $R \leftarrow \frac{R+qP}{2^w}$
  $c \leftarrow 0$
  $(c, \cdot) \leftarrow q \times p[0] + r[0] + c$
  for $j = 1$ **to** $m-1$ **do**
    $(c, r[j-1]) \leftarrow q \times p[j] + r[j] + c$
  end for
  $(c, r[m-1]) \leftarrow s + c$
  $s \leftarrow t$
▷ End Step $R \leftarrow \frac{R+qP}{2^w}$
▷ Invariant: $R2^{(i+1)w} = (a[i]2^{iw} + \cdots + a[0])B \bmod P$
▷ Invariant: $R < P + B$
end for

▷ **End Main Loop**
▷ **Reduction step**
if $s = 1$ **then**
  $(c, S) \leftarrow \text{SUB}(R, P, m)$
  return S
else
  if $R \geq P$ **then**
    $(c, S) \leftarrow \text{SUB}(R, P, m)$
    return S
  else
    return R
  end if
end if

When $P, \tilde{p}, m$ are implicit, we simply denote montMUL(A, B) the Montgomery multiplication of A, B.

Like the modular addition, there is a conditional step depending on the value at the end. By adding extra words to the manipulated integers, the final subtraction condition can be avoided [Wal99, HQ00]. However, this solution is costly. For a constant time Montgomery multiplication, one can perform a subtraction whatever the value after the main loop, as for the modular addition. This can be done by replacing the reduction step by:

$c \leftarrow s$
$(d, S) \leftarrow \text{SUB}(R, P, m)$
if $d \oplus c = 0$ **then**
  return S
else
  return R

The justification of the correctness of the reduction step is the same as the modular addition (Algorithm 6).

To avoid the resetting data of line 1, the first iteration ($i = 0$), line 9 can be replaced by $(c, r[j]) \leftarrow a[j] \times b[j] + c$. Also, it is possible to improve the algorithm if $A = B$ (montSQR).

Denote $C = A \times B \bmod P$ the product of two integers A, B modulo P. The Montgomery multiplication of A, B returns $C \times 2^{-wm} \bmod P$. Let $A' = A2^{wm} \bmod P$ and $B' = B2^{wm} \bmod P$. The Montgomery multiplication of A', B' returns

$$\begin{aligned} \text{montMUL}(A', B') &= A2^{wm} \times B2^{wm} \times 2^{-wm} \quad \bmod P \\ &= C2^{wm} \quad \bmod P \end{aligned}$$

Denote $C' = C2^{wm} \bmod P$. A', B', C' are called the Montgomery representations of A, B, C respectively. The Montgomery multiplication of integers in their Montgomery representations returns the Montgomery representation of the product. At the beginning of ECC applications, Montgomery representations of the manipulated integers are pre-computed. Trivially, the modular addition of integers in their Montgomery representation returns the Montgomery representation of the sum:

$$\begin{aligned} A' + B' &= A2^{wm} + B2^{wm} \quad \bmod P \\ &= (A + B)2^{wm} \quad \bmod P \end{aligned}$$

Obviously, the same holds for modular subtraction. Integers in their Montgomery representation are then manipulated in a transparent manner, as if they were in the regular form. At the end of an ECC application, the correct values are easily recovered using $A = \text{montMUL}(A', 1)$. Algorithm 8 can be optimized if one of the operand is 1.

The pre-computations of the Montgomery representation of the integers is the main drawback of the Montgomery multiplication. The pre-computation of the Montgomery representation of A can be done by computing $A' = \text{montMUL}(A, R)$ with $R = 2^{2wm} \bmod P$. One way to compute R is to perform:

$R \leftarrow \text{montMUL}(1, 1)$ ▷ $R = 2^{-wm}$
$R \leftarrow \text{montMUL}(R, 1)$ ▷ $R = 2^{-2wm}$
$R \leftarrow \text{mINV}(R, P, m)$ ▷ $R = 2^{2wm}$ (modular inversion described below)

2.5 Modular Inversion

Modular inversions are costly and are avoided as much as possible, using projective or Jacobian coordinates. It is estimated that the cost of a modular inversion is approximatively 100 times the cost of a modular multiplication [Ver12, §1.1.1.6], which is the case for our implementation. Modular inversion is however required at the end of the ECSM for the conversion from projective to affine coordinates, for some cryptographic protocols or for some countermeasures. We give the unsigned version of the binary inversion algorithm of [HMV03, Algorithm 2.22].

Algorithm 9 Euclidean Modular Inversion (mINV)

Input: A, P, m such that $gcd(A, P) = 1$, P is odd and $A < 2^{wm}, P < 2^{wm}$
Output: $A^{-1} \bmod P$

```
U ← A
V ← P
X ← 1
Y ← 0

while U ≠ 1 and V ≠ 1 do
    while U is even do
        c ← 0
        U ← (U >> 1)                                   ▷ right shift
        if X is odd then                               ▷ perform X ← (X + P)/2 with the following
            (c, X) ← ADD(X, P, m)
            X ← (X >> 1)
            if c = 1 then X ← X + 2^(wm−1)             ▷ set the most significant bit to 1
    while V is even do
        c ← 0
        V ← (V >> 1)
        if Y is odd then
            (c, Y) ← ADD(Y, P, m)
            Y ← (Y >> 1)
            if c = 1 then Y ← Y + 2^(wm−1)             ▷ set the most significant bit to 1

    if U ≥ V then
        (c, U) ← SUB(U, V, m)
        X ← mSUB(X, Y, P, m)
    else
        (c, V) ← SUB(V, U, m)
        Y ← mSUB(Y, X, P, m)
if U = 1 then
    return X
else
    return Y
```

2.6 Cost of Arithmetic Operations

In Part II, we are interested on the theoretical cost of the countermeasures against physical attacks. For some applications or countermeasures, it is also necessary to have an access to long integer arithmetic such as addition, subtraction, multiplication, square and Euclidean division with remainder. Since the real cost depends on the architecture and on the size of the integers, we choose the following notation:

- ADD_n: execution time of an addition or a subtraction[2] of n-bit integers,
- SQR_n: execution time of a square of a n-bit integer,
- MUL_n: execution time of a multiplication of n-bit integers,
- DIV_n: execution time of a division of n-bit integers,
- mADD_n: execution time of a modular addition or subtraction[2] of n-bit integers,
- mSQR_n: execution time of a modular square of a n-bit integer[3],
- mMUL_n: execution time of a modular multiplication of n-bit integers[3],
- mINV_n: execution time of a modular inversion of a n-bit integer[3].

[2]The cost of an addition and subtraction are rarely significantly different.
[3]The modulus is a n-bit integer as well.

Chapter 3

Elliptic Curve Arithmetic

The formulæ given in Section 1.4 (ECADD and ECDBL) may be used to perform addition and doubling. During the past years, many different formulæ have been proposed for different elliptic curves and different coordinates. So far, standardized curves for cryptographic applications are given in the short Weierstraß equation. We thus restrict to these elliptic curves where the Jacobian coordinates are the most efficient. The choice of the formulæ is made depending on the ECSM used, on some physical security requirements or on the memory available.

We give in Section 3.1 the most commonly used formulæ for elliptic curves in the short Weierstraß equation. It is essential for describing and understanding some physical attacks. Moreover, some countermeasures consist in slightly modifying the formulæ. For a more detailed explanation on the formulæ and on their cost, one can refer to the synthesis given in [BL04] and [Ver12, Chapter 1]. In addition, we give the memory required. Section 3.2 summarizes the cost in number of field operations and the memory required.

3.1 Elliptic Curve Formulæ

This section gives the most efficient formulæ for elliptic curves over $\mathbb{F}_p$, with p a n-bit prime integer, in the short Weierstraß equation. We distinguish classical operations (addition and doubling) from the co-Z formulæ where more refined operations are presented: the addition and update, and the conjugate addition and update formulæ.

The number of registers for each formula is given without considering input and output points. Only the number of extra temporary registers is taken into account. MEM_l denotes a memory block of l bits. We can save some memory with additional field additions and subtractions as in [GJM+11]. We give the memory required for an optimal number of finite field operations.

This section gives high level algorithm of elliptic curve operations. The same algorithms with register allocation are given in appendix, only if they are useful for the description of some attacks or countermeasures.

For all formulæ, the points are in the Jacobian coordinates system, and they are different from the point at infinity.

3.1.1 Classical Formulæ

Algorithm 10 ECADD

Input: $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$ with $Q \neq \pm P$
Output: $P + Q$
$A \leftarrow X_1 Z_2^2;\ B \leftarrow X_2 Z_1^2;\ C \leftarrow Y_1 Z_2^3;\ D \leftarrow Y_2 Z_1^3$
$E \leftarrow B - A;\ F \leftarrow D - C$
$X_3 \leftarrow F^2 - E^3 - 2AE^2$
$Y_3 \leftarrow F(AE^2 - X_3) - CE^3$
$Z_3 \leftarrow Z_1 Z_2 E$
return (X_3, Y_3, Z_3)

If Q is in affine coordinates ($Z_2 = 1$), one can save four multiplications and one square [CMO98]. It is called mixed addition (mECADD). If Z_2^2 and Z_2^3 are pre-computed, one multiplication and one square are saved [CC86]. It is called re-addition (reECADD).
Cost (ECADD): 12 mMUL_n, 4 mSQR_n, 7 mADD_n, 3 MEM_n
Cost (mECADD): 8 mMUL_n, 3 mSQR_n, 7 mADD_n, 3 MEM_n
Cost (reECADD): 11 mMUL_n, 3 mSQR_n, 7 mADD_n, 3 MEM_n

Algorithm 11 ECDBL

Input: $P = (X_1, Y_1, Z_1)$ with $ord(P) \neq 2$, elliptic curve parameter a
Output: $2P$
$A \leftarrow 2Y_1^2$
$B \leftarrow 2AX_1;\ C \leftarrow 3X_1^2 + aZ_1^4;\ D \leftarrow 2A^2$
$X_3 \leftarrow C^2 - 2B$
$Y_3 \leftarrow C(B - X_3) - D$
$Z_3 \leftarrow 2Y_1 Z_1$
return (X_3, Y_3, Z_3)

If (X_3, Y_3, Z_3) is used later for re-addition, computing Z_3^2 and Z_3^3 needs one extra square and one extra multiplication (reECDBL). If $W_1 = aZ_1^4$ is pre-computed, two squares are saved, and it needs one extra addition[1] [CMO98]. It is called the modified Jacobian coordinates (modECDBL). The use of both modified coordinates and re-addition is also given (mod-reECDBL).
Cost (ECDBL): 4 mMUL_n, 6 mSQR_n, 11 mADD_n, 3 MEM_n
Cost (reECDBL): 5 mMUL_n, 7 mSQR_n, 11 mADD_n, 3 MEM_n
Cost (modECDBL): 4 mMUL_n, 4 mSQR_n, 12 mADD_n, 3 MEM_n
Cost (mod-reECDBL): 5 mMUL_n, 5 mSQR_n, 12 mADD_n, 3 MEM_n

Remark 3.1. In the standardized elliptic curves recommended in [FIPS186-3], the parameter a of the curve is equal to -3. This makes it possible to perform a faster computation of $C = 3X_1^2 + aZ_1^4$ in this way: $C \leftarrow 3(X_1 + Z_1^2)(X_1 - Z_1^2)$. This saves two squares and costs an extra subtraction over ECDBL. The formula is called *fast doubling*.

3.1.2 Co-Z Formulæ

An ECSM is generally based on addition and doubling formulæ of points. Meloni shows that addition of two points of an elliptic curve is more efficient if they share the same Z-coordinate [Mel07].

[1] In fact, two squares and one multiplication are saved. An extra addition and an extra multiplication are needed for the computation of $W_3 = aZ_3^4 = 2DW_1$.

He brought a new formula, which was called later co-Z addition and update (ZADDU) in [GJM10]. This formula alone is enough to perform an ECSM with addition chains and Zeckendorf representation [Mel07].

Algorithm 12 co-Z addition and update (ZADDU) [Mel07]

Input: $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z)$ with $Q \neq \pm P$
Output: (R, S) with $R = P + Q$ and $S = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z)$ with $\lambda = X_1 - X_2$
$C \leftarrow (X_1 - X_2)^2$
$W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C;\ Z_3 \leftarrow Z(X_1 - X_2)$
$D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_3 \leftarrow D - W_1 - W_2$
$Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
$X_4 \leftarrow W_1$
$Y_4 \leftarrow A_1$
return $((X_3, Y_3, Z_3), (X_4, Y_4, Z_3))$

The computation of the Z coordinate is not necessary if the Montgomery Ladder is used for the ECSM [GJM+11]. It is called (X, Y)-only co-Z addition and update (ZADDU'). One multiplication is saved. The final Z coordinate is recovered at the end of the ECSM.
Cost (ZADDU): 5 mMUL_n, 2 mSQR_n, 7 mADD_n, 2 MEM_n
Cost (ZADDU'): 4 mMUL_n, 2 mSQR_n, 7 mADD_n, 2 MEM_n

Meloni's formula was extended. The conjugate co-Z addition (ZADDC) was proposed in [GJM10]. It consists in computing the points $P + Q$ and $P - Q$ within the same formula. In [GJM10, HJS11, GJM+11], with both ZADDU and ZADDC, the authors show that co-Z formulæ might be usable with classical ECSM algorithms such as the Right-to-Left signed-digit method, the Montgomery Powering Ladder [JY02], or the Joye's double-add method [Joy07].

Algorithm 13 conjugate co-Z addition (ZADDC) [GJM10]

Input: $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z)$ with $Q \neq \pm P$
Output: (R, S) with $R = P + Q$, $S = P - Q$
$C \leftarrow (X_1 - X_2)^2$
$W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C;\ Z_3 \leftarrow Z(X_1 - X_2)$
$D_1 \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_3 \leftarrow D_1 - W_1 - W_2$
$Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
$D_2 \leftarrow (Y_1 + Y_2)^2$
$X_4 \leftarrow D_2 - W_1 - W_2$
$Y_4 \leftarrow (Y_1 + Y_2)(W_1 - X_4) - A_1$
return $((X_3, Y_3, Z_3), (X_4, Y_4, Z_3))$

The computation of the Z coordinate is not necessary if the Montgomery Ladder is used [GJM+11]. It is called (X, Y)-only co-Z conjugate addition (ZADDC'). One multiplication is saved. The final Z coordinate is recovered at the end of the ECSM.
Cost (ZADDC): 6 mMUL_n, 3 mSQR_n, 11 mADD_n, 3 MEM_n
Cost (ZADDC'): 5 mMUL_n, 3 mSQR_n, 11 mADD_n, 3 MEM_n

3.2 Cost Summary

Table 3.1 gives the cost of the different formulæ of this section. The memory cost is the number of temporary registers needed for the formula, excluding input and output memory blocks.

Point doubling	
ECDBL	4 mMUL_n, 6 mSQR_n, 11 mADD_n, 3 MEM_n
reECDBL	5 mMUL_n, 7 mSQR_n, 11 mADD_n, 3 MEM_n
modECDBL	4 mMUL_n, 4 mSQR_n, 12 mADD_n, 3 MEM_n
mod-reECDBL	5 mMUL_n, 5 mSQR_n, 12 mADD_n, 3 MEM_n
Points addition	
ECADD	12 mMUL_n, 4 mSQR_n, 7 mADD_n, 3 MEM_n
mECADD	8 mMUL_n, 3 mSQR_n, 7 mADD_n, 3 MEM_n
reECADD	11 mMUL_n, 3 mSQR_n, 7 mADD_n, 3 MEM_n
ZADDU	5 mMUL_n, 2 mSQR_n, 7 mADD_n, 2 MEM_n
ZADDU'	4 mMUL_n, 2 mSQR_n, 7 mADD_n, 2 MEM_n
Points conjugate addition	
ZADDC	6 mMUL_n, 3 mSQR_n, 11 mADD_n, 3 MEM_n
ZADDC'	5 mMUL_n, 3 mSQR_n, 11 mADD_n, 3 MEM_n

Table 3.1: Cost of the most commonly used elliptic curve formulæ for a n-bit prime modulus.

We use the following notation to quantify the different ECSMs described in the next chapter and the cost of some countermeasures increasing the number of elliptic curve operations:

- ECADD_n: execution time of an elliptic curve points addition with a modulus of size n,
- ECDBL_n: execution time of an elliptic curve point doubling with a modulus of size n,
- C-ECADD_n: execution time of an elliptic curve points conjugate addition with a modulus of size n.

Chapter 4

Elliptic Curve Scalar Multiplication

In ECC applications, one has to compute scalar multiplications (ECSMs), *i.e.* compute

$$[k]P = \underbrace{P + \cdots + P}_{k \text{ times}},$$

given a point P and an integer k. The choice of an ECSM algorithm depends on the memory constraints and on the physical security requirements.

ECSMs are generally performed with a loop scanning of the bits of the scalar. In fact, the ECSMs can be compared to the modular exponentiation methods used in RSA and systems based on the multiplicative group of a finite field: an addition of points is replaced by a multiplication, and a doubling is replaced by a square. In addition, in elliptic curves we can take advantage of the useful property that the inverse element of the point $P = (x, y)$ is $-P = (x, -y)$. This operation is almost free; hence, it can be done on the fly. For modular exponentiations, the analogy would be computing modular inverses, which are very costly.

The ECSMs are generally organized depending of the regularity. An ECSM is said to be *regular* if, at each iteration, the same elliptic curve operations are performed whatever the value of the scalar. Unregular ECSMs are generally more efficient. However, we will see in Section 8.2 that unregular ECSMs can be vulnerable to the Simple Side-Channel Analysis. We will also describe some related countermeasures.

The ECSMs described in this section can use different formulæ given in the previous section, depending on the performance and on the security requirements. We use the notation "$\cdot + \cdot$" for an elliptic curve addition and "$2\cdot$" for an elliptic curve doubling. We evaluate the cost of the ECSM using the notations of Section 3.2 where n is the bit-length of the prime modulus. A more detailed comparison on the cost of the ECSMs can be found in [Ver12, Chapter 1]. In addition, we give the memory required. A summary of the cost of the ECSMs is given in Section 4.3.

4.1 Unregular ECSMs

We give in Algorithms 14, 15, 16, 17, 18, the most commonly used unregular ECSMs.

Algorithm 14 Left-to-Right Double-and-Add

Input: a point P and an integer $k = (1, k_{n-2}, \ldots, k_0)_2$
Output: $[k]P$
  $R_0 \leftarrow P$
  for $i = n-2$ **downto** 0 **do**
    $R_0 \leftarrow 2R_0$ ▷ $R_0 = [(k_{n-1}, \ldots, k_{i+1}, 0)_2]P$
    if $k_i = 1$ **then** $R_0 \leftarrow R_0 + P$ ▷ $R_0 = [(k_{n-1}, \ldots, k_{i+1}, k_i)_2]P$
  return R_0

Cost: $\frac{n}{2}$ ECADD$_n$ + n ECDBL$_n$

Algorithm 15 Right-to-Left Double-and-Add

Input: $k = (k_{n-1}, \ldots, k_1, 1)_2, P$
Output: $[k]P$
  $R_0 \leftarrow P$
  $R_1 \leftarrow 2P$
  for $i = 1$ **to** $n-1$ **do**
    if $k_i = 1$ **then** $R_0 \leftarrow R_0 + R_1$ ▷ $R_0 = [(k_i, \ldots, k_0)_2]P$
    $R_1 \leftarrow 2R_1$ ▷ $R_1 = [2^{i+1}]P$
  return R_0

Cost: $\frac{n}{2}$ ECADD$_n$ + n ECDBL$_n$

Computing the inverse element of a point is almost free: if $P = (x, y)$, then $-P = (x, -y)$. Using a signed digit representation of the scalar can speed up some algorithms. The *Non-Adjacent Form* (NAF) is suitable. The following definitions and properties of the NAF can be found in [HMV03, Section 3.3].

Definition 4.1. A *non-adjacent form* (NAF) of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{-1, 0, 1\}$, $k_{l-1} \neq 0$, and no two consecutive digits k_i are nonzero. The *length* of the NAF is l.

Theorem 4.2. [HMV03, Theorem 3.29]. *Let k be a positive integer.*

- *k has a unique NAF denoted NAF(k) or $(k_{l-1}, \ldots, k_0)_{NAF}$.*
- *NAF(k) has the fewest nonzero digits of any signed digit representation of k.*
- *l is at most one more than the length of the binary representation of k.*
- *The average density of nonzero digits among all NAFs of length n is approximatively $1/3$.*

Algorithm 16 Left-to-Right sliding window NAF scalar multiplication

Input: $k = (1, k_{l-2}, \ldots, k_0)_{\text{NAF}}, v \geq 2, P$
Output: $[k]P$
  $m \leftarrow 2(2^v - (-1)^v)/3 - 1$
  $Q \leftarrow P$
  for $i = 1$ **to** m **by** 2 **do**
    $P_i \leftarrow [i]P$
  $i \leftarrow l - 2$
  while $i \geq 0$ **do**
    if $k_i = 0$ **then**
      $Q \leftarrow 2Q$
      $i \leftarrow i - 1$
    else
      $s \leftarrow \max(i - v + 1, 0)$
      while $k_s = 0$ **do**
        $s \leftarrow s + 1$
      $u \leftarrow (k_i, \ldots, k_s)_{\text{NAF}}$
      for $j = 1$ **to** $i - s + 1$ **do**
        $Q \leftarrow 2Q$
      if $u > 0$ **then**
        $Q \leftarrow Q + P_u$
      if $u < 0$ **then**
        $Q \leftarrow Q - P_{-u}$
      $i \leftarrow s - 1$
  return Q

Cost: $\left(\frac{l}{v+f(v)} + \frac{2^v-(-1)^v}{3} - 1\right)$ $\text{ECADD}_n + (l+1)$ ECDBL_n, with $f(v) = \frac{4}{3} - \frac{(-1)^v}{3\times 2^{v-2}}$

The NAF can be generalized to larger digits.

Definition 4.3. Let $v \geq 2$ be a positive integer. A *width-v NAF* of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where each nonzero coefficient is odd, $-2^{v-1} < k_i < 2^{v-1}$, $k_{l-1} \neq 0$, and at most one of any v consecutive digits is nonzero. The *length* of the NAF is l.

Theorem 4.4. [HMV03, Theorem 3.33]. *Let k be a positive integer.*

- *k has a unique width-v NAF denoted $NAF_v(k)$.*
- *$NAF_2(k) = NAF(k)$.*
- *l is at most one more than the length of the binary representation of k.*
- *The average density of nonzero digits among all width-v NAFs of length l is approximatively $1/(v+1)$.*

The following algorithm computes the width-v NAF representation of the scalar on the fly.

Algorithm 17 Right-to-Left sliding window width-v NAF scalar multiplication

Input: $k = (k_{n-1}, \ldots, k_0)_2, v \geq 2, P$
Output: $[k]P$
  $m \leftarrow 2^{v-1} - 1$
  $R \leftarrow P$
  $Q_1, Q_3, \ldots, Q_m \leftarrow \mathcal{O}$
  while $k \geq 1$ **do**
    if $k_0 = 1$ **then**
      $u \leftarrow (k_{v-1}, \ldots, k_0)$
      if $u \geq 2^{v-1}$ **then** $u \leftarrow u - 2^v$
      if $u > 0$ **then**
        $Q_u \leftarrow Q_u + R$
      if $u < 0$ **then**
        $Q_{-u} \leftarrow Q_{-u} - R$
      $k \leftarrow k - u$
    $R \leftarrow 2R$
    $k \leftarrow k/2$
  for $i = 3$ **to** m **by** 2 **do**
    $Q_1 \leftarrow Q_1 + [i]Q_i$
  return Q_1

Cost: $\left(\frac{n}{v+1} + 2^{2v-4} - 1\right)$ $\text{ECADD}_n + n$ ECDBL_n

The Shamir's trick computes $[k]P + [d]S$ with a single loop scanning.

Algorithm 18 Shamir's trick [Str64]

Input: $k = (k_{n-1}, \ldots, k_0)_2$, $d = (d_{n-1}, \ldots, d_0)_2$ with $(k_{n-1}, d_{n-1}) \neq (0, 0)$, P, S
Output: $[k]P + [d]S$
  $R_1 \leftarrow P$; $R_2 \leftarrow S$; $R_3 \leftarrow P + S$
  $c \leftarrow 2d_{n-1} + k_{n-1}$; $R_0 \leftarrow R_c$
  for $i = n - 2$ **downto** 0 **do**
    $R_0 \leftarrow 2R_0$ ▷ $R_0 = [(k_{n-1}, \ldots, k_{i+1}, 0)_2]P + [(d_{n-1}, \ldots, d_{i+1}, 0)_2]S$
    $c \leftarrow 2d_i + k_i$
    if $c \neq 0$ **then** $R_0 \leftarrow R_0 + R_c$
    ▷ $R_0 = [(k_{n-1}, \ldots, k_{i+1}, k_i)_2]P + [(d_{n-1}, \ldots, d_{i+1}, d_i)_2]S$
  return R_0

Cost: $\frac{3n}{4}$ $\text{ECADD}_n + n$ ECDBL_n

4.2 Regular ECSMs

We give in Algorithms 19, 20, 21, 22, 23, 24, 25 the most commonly used regular ECSMs.

Algorithm 19 Left-to-Right Double-and-Add always [Cor99, §3.1]

Input: $k = (1, k_{n-2}, \ldots, k_0)_2$, P
Output: $[k]P$
$R_0 \leftarrow P, R_1 \leftarrow 2P$
for $i = n-2$ **downto** 0 **do**
$R_0 \leftarrow 2R_0$
$R_{1-k_i} \leftarrow R_{1-k_i} + P$
return R_0

Cost: n ECADD$_n$ + n ECDBL$_n$

Algorithm 20 Right-to-Left Double-and-Add always

Input: $k = (k_{n-1}, \ldots, k_1, 1)_2, P$
Output: $[k]P$
$R_0 \leftarrow P$
$R_1 \leftarrow P$ ▷ point for dummy operations
$R_2 \leftarrow 2P$
for $i = 1$ **to** $n-1$ **do**
$R_{1-k_i} \leftarrow R_{1-k_i} + R_2$ ▷ $R_0 = [(k_i, \ldots, k_0)_2]P$ and $R_1 = [(\bar{k}_i, \ldots, k_0)_2]P$
$R_2 \leftarrow 2R_2$ ▷ $R_2 = [2^{i+1}]P$
return R_0

Cost: n ECADD$_n$ + n ECDBL$_n$

Algorithm 21 Montgomery Ladder [JY02]

Input: $k = (1, k_{n-2}, \ldots, k_0)_2, P$
Output: $[k]P$
1: $R_0 \leftarrow P, R_1 \leftarrow 2P$
2: **for** $i = n-2$ **downto** 0 **do**
3: $R_{1-k_i} \leftarrow R_0 + R_1$
4: $R_{k_i} \leftarrow 2R_{k_i}$ ▷ $R_0 = [(k_{n-1}, \ldots, k_i)_2]P$ and $R_1 = R_0 + P$
5:
6: **return** R_0

Cost: n ECADD$_n$ + n ECDBL$_n$

The Montgomery Ladder, adapted with the co-Z formulæ, is given below.

Algorithm 22 Montgomery Ladder with co-Z formulæ [GJM10]

Input: $k = (1, k_{n-2}, \ldots, k_0)_2, P$
Output: $[k]P$
  $R_0 \leftarrow P, R_1 \leftarrow 2P$
  for $i = n-2$ **downto** 0 **do**
    $(R_{1-k_i}, R_{k_i}) \leftarrow \text{ZADDC}(R_{k_i}, R_{1-k_i})$
    $(R_{k_i}, R_{1-k_i}) \leftarrow \text{ZADDU}(R_{1-k_i}, R_{k_i})$ ▷ $R_0 = [(k_{n-1}, \ldots, k_i)_2]P$ and $R_1 = R_0 + P$
  return R_0

Cost: n C-ECADD$_n$ + n ECADD$_n$

The following algorithm is the regular version of Algorithm 18 by adding a dummy operation.

Algorithm 23 Regular Shamir's trick [CJ03]

Input: $k = (k_{n-1}, \ldots, k_0)_2$, $d = (d_{n-1}, \ldots, d_0)_2$ with $(k_{n-1}, d_{n-1}) \neq (0, 0)$, P, S
Output: $[k]P + [d]S$
  $R_1 \leftarrow P$; $R_2 \leftarrow S$; $R_3 \leftarrow P + S$
  $R_4 \leftarrow P + S$ ▷ R_4 is used for dummy operations
  $c \leftarrow 2d_{n-1} + k_{n-1}$; $R_0 \leftarrow R_c$
  for $i = n-2$ **downto** 0 **do**
    $R_0 \leftarrow 2R_0$ ▷ $R_0 = [(k_{n-1}, \ldots, k_{i+1}, 0)_2]P + [(d_{n-1}, \ldots, d_{i+1}, 0)_2]S$
    $b \leftarrow \neg(k_i \vee d_i)$; $c \leftarrow 2d_i + k_i$
    $R_{4b} \leftarrow R_{4b} + R_c$ ▷ $R_0 = [(k_{n-1}, \ldots, k_i)_2]P + [(d_{n-1}, \ldots, d_i)_2]S$
  return R_0

Cost: n ECADD$_n$ + n ECDBL$_n$

Algorithm 24 Binary Random Initial Point (BRIP) [MMM04]

Input: $k = (k_{n-1}, \ldots, k_0)_2, P$
Output: $[k]P$
  $S \leftarrow random\_point()$
  $R_0 \leftarrow S$, $R_1 \leftarrow -S, R_2 = P - S$
  for $i = n-1$ **downto** 0 **do**
    $R_0 \leftarrow 2R_0$ ▷ $R_0 = [(k_{n-1}, \ldots, k_{i+1}, 0)_2]P + [2]S$
    $R_0 \leftarrow R_0 + R_{1+k_i}$ ▷ $R_0 = [(k_{n-1}, \ldots, k_i)_2]P + S$
  return $R_0 + R_1$

Cost: n ECADD$_n$ + n ECDBL$_n$

The following algorithm was introduced for RSA applications. We adapted it for elliptic curves. We removed the final coherence check which will be discussed in Section 8.18.

Algorithm 25 Blinded Right-to-Left Double-and-Add always [BHT09]

Input: $k = (k_{n-1}, \ldots, k_0)_2, P$
Output: $[k]P$
  $S \leftarrow random\_point()$
  $R_0 \leftarrow S,\ R_1 \leftarrow -S, R_2 = P$
  for $i = 0$ **to** $n-1$ **do**
    $R_{1-k_i} \leftarrow R_{1-k_i} + R_2$ ▷ $R_0 = [(k_i, \ldots, k_0)_2]P + S$ and $R_1 = [(\bar{k_i}, \ldots, \bar{k_0})_2]P - S$
    $R_2 \leftarrow 2R_2$ ▷ $R_2 = [2^{i+1}]P$
  return $R_0 - S$

Cost: n ECADD$_n$ + n ECDBL$_n$

4.3 Cost Summary

Table 4.1 gives the cost of the different ECSMs without any countermeasure. For each ECSM, the more suitable formulæ from Chapter 3 are chosen regarding the efficiency. The memory cost is calculated from the memory blocks required to store all involved points, the intermediate variables for elliptic curve operations, and the curve parameters a, p (b is not used).

Remark 4.5. The number of field operations is given in average per bit of scalar. Obviously, the number is exact for regular ECSMs since the operations do not depend on the scalar.

Remark 4.6. The length of the NAF is at most one more than the length of the binary representation of the scalar. In ECC applications, the prime modulus is at least a 256-bit integers. Therefore, we can neglect the extra possible digit when the scalar in expressed in the NAF representation. In Table 4.1, we expose the cost of the ECSMs without distinguishing the representation of the scalar.

Some countermeasures against physical attacks increase the number of iterations of the ECSM or increase the number of ECSMs. We denote by ECSM$_{l,n}$ the execution time of an ECSM with a l-bit scalar and a n-bit modulus.

Unregular ECSMs	
L-to-R Double-Add ECDBL and mECADD	8 mMUL_n, 7.5 mSQR_n, 14.5 mADD_n, 8 MEM_n
R-to-L Double-Add modECDBL and ECADD	10 mMUL_n, 6 mSQR_n, 15.5 mADD_n, 10 MEM_n
L-to-R window NAF ($v = 2$) ECDBL and mECADD	6.7 mMUL_n, 5 mSQR_n, 13.4 mADD_n, 8 MEM_n
L-to-R window NAF ($v = 3$) ECDBL and mECADD	5.8 mMUL_n, 4.7 mSQR_n, 12.6 mADD_n, 12 MEM_n
L-to-R window NAF ($v = 4$) ECDBL and mECADD	5.6 mMUL_n, 4.6 mSQR_n, 12.4 mADD_n, 16 MEM_n
R-to-L window NAF ($v = 2$) modECDBL and ECADD	8 mMUL_n, 5.4 mSQR_n, 14.4 mADD_n, 10 MEM_n
R-to- L window NAF ($v = 3$) modECDBL and ECADD	7 mMUL_n, 5 mSQR_n, 13.8 mADD_n, 13 MEM_n
R-to-L window NAF ($v = 4$) modECDBL and ECADD	6.4 mMUL_n, 4.8 mSQR_n, 13.4 mADD_n, 19 MEM_n
Shamir's trick ECDBL and mECADD	9 mMUL_n, 6.75 mSQR_n, 13.5 mADD_n, 12 MEM_n
Regular ECSMs	
L-to-R Double-Add always ECDBL and mECADD	12 mMUL_n, 9 mSQR_n, 18 mADD_n, 11 MEM_n
R-to-L Double-Add always modECDBL and ECADD	18 mMUL_n, 8 mSQR_n, 19 mADD_n, 13 MEM_n
Montgomery Ladder ZADDU' and ZADDC'	9 mMUL_n, 5 mSQR_n, 18 mADD_n, 9 MEM_n
Regular Shamir's trick ECDBL and mECADD	12 mMUL_n, 9 mSQR_n, 18 mADD_n, 13 MEM_n
BRIP ECDBL and mECADD	12 mMUL_n, 9 mSQR_n, 18 mADD_n, 12 MEM_n
Blinded R-to-L Double-Add always modECDBL and ECADD	18 mMUL_n, 8 mSQR_n, 19 mADD_n, 13 MEM_n

Table 4.1: Average cost per bit of scalar for the most commonly used ECSMs for a n-bit prime modulus

Chapter 5

Cryptographic Protocol

Given the following curve parameters:

- E, an elliptic curve over a prime field $\mathbb{F}_p$,
- G, a generator of a subgroup of E of order t,

a private key is expressed by an integer d randomly chosen in $\{1, t-1\}$. The corresponding public key is $P = [d]G$.

Such a key pair is involved in cryptographic protocols such as the signature scheme ECDSA (Section 5.1), the key agreement protocol ECDH (Section 28) or the encryption scheme EC-ELGAMAL (Section 5.3).

5.1 Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a signature scheme. It has been standardized in [ANSI X9.62].

Algorithm 26 ECDSA Signature

Input: private key d, an encoded integer $m \in [0, p-1]$ representing a message
Output: Signature (r, s)

1: $k \xleftarrow{\mathcal{R}} \{1, \ldots, t-1\}$
2: $Q \leftarrow [k]G$
3: $r \leftarrow x_Q \bmod t$
4: **if** $r = 0$ **then**
5: **go to** line 1
6: $k_{inv} \leftarrow k^{-1} \bmod t$
7: $s \leftarrow k_{inv}(dr + m) \bmod t$
8: **if** $s = 0$ **then**
9: **go to** line 1
10: **return** (r, s)

Algorithm 27 ECDSA Verification

Input: public key P, an encoded integer $m \in [0, p-1]$ representing a message, signature (r, s)
Output: true or false

$s_{inv} \leftarrow s^{-1} \bmod t$
$u_1 \leftarrow s_{inv} \times m \bmod t$
$u_2 \leftarrow s_{inv} \times r \bmod t$
$Q \leftarrow [u_1]G + [u_2]P$
$v \leftarrow x_Q \bmod t$
if $v = r$ **then**
    return true
else
    return false

The Shamir's trick (Algorithm 18) is suitable for the verification procedure.

5.2 Elliptic Curve Diffie Hellman

The Elliptic Curve Diffie Hellman (ECDH) cryptographic scheme is a key agreement between two entities. It was standardized in [ANSI X9.63]. The procedure enables to share a secret data from the private key of the first entity and the public key of the second entity.

Algorithm 28 ECDH

Input: A's private key d_A, B's public key P_B
Output: Secret Point S

$S \leftarrow [d_A]P_B$
if $S = \mathcal{O}$ **then**
    return ERROR
return S

The entity B does the same with his own private key d_B and A's public key P_A. Both respective calculations give the point $S = [d_A d_B]G$. A and B now share a secret data. ECDH can be used with ephemeral or static keys.

5.3 Elliptic Curve ElGamal

The Elliptic Curve ElGamal (EC-ELGAMAL) is a cipher scheme.

Algorithm 29 EC-ELGAMAL Encryption

Input: Public Key P, an encoded integer $m \in [0, p-1]$ representing a message
Output: (x_1, y_1, c)

$k \xleftarrow{\mathcal{R}} \{1, \ldots, t-1\}$
$(x_1, y_1) \leftarrow [k]G$
$(x_2, y_2) \leftarrow [k]P$
$c \leftarrow x_2 + m \bmod p$
return (x_1, y_1, c)

Algorithm 30 EC-ELGAMAL Decryption

Input: Private Key d, an encrypted message (x_1, y_1, c)
Output: an encoded integer $m' \in [0, p-1]$ representing a message
$(x'_2, y'_2) \leftarrow [d](x_1, y_1)$
$m' \leftarrow c - x'_2 \bmod p$
return m'

Chapter 6

ECC Security

The security of ECC in the black box model relies on the hardness of one of the following problems:

- the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**, that is the computation of k given P and $Q = [k]P$,
- the **Elliptic Curve Computational Diffie Hellman**, that is the computation of $[k_1k_2]P$ given P, $Q_1 = [k_1]P$ and $Q_2 = [k_2]P$,
- the **Elliptic Curve Decisional Diffie Hellman**, that is, given P, $Q_1 = [k_1]P$, $Q_2 = [k_2]P$ and $Q_3 = [k_3]P$, assess if $Q_3 = [k_1k_2]P$.

The Elliptic Curve Computational Diffie Hellman and the Elliptic Curve Decisional Diffie Hellman are trivially solvable if the ECDLP is. The best known algorithms to solve the ECDLP are the Pollard's rho [Pol78] and the Baby-step Giant-Step [Sha71] methods. They both have a complexity of $\mathcal{O}(\sqrt{t})$ where $t = ord(P)$. If l is the security parameter of the cryptographic application (e.g. 128 or 256), the recommended elliptic curve shall have a point P of order $ord(P) \approx 2^{2l}$. Although there is no concrete proof that the ECDLP cannot be solvable with better algorithms, this problem has been accepted by the cryptographic community to guarantee the security of ECC.

Note that, so far, no sub-exponential algorithm is known to solve the ECDLP. This is not the case for integers factorization (which immediately breaks RSA) and discrete logarithms on the multiplicative group of a finite field.

Remark 6.1. In particular elliptic curves, such as anomalous curves, supersingular curves, or other curves [BSS99, Chapter III], better methods than the Pollard's rho and the Baby-step Giant-Step are known for solving the ECDLP. Elliptic curves for cryptographic applications are chosen with good care so they do not have any of those particularities, and the Pollard's rho or the Baby-step Giant-Step methods are the best known methods to solve the ECDLP.

Generally, physical attacks aim at recovering the scalar or breaking the protocol with totally different methods, without solving the ECDLP. Fortunately, several methods exist to thwart them. This is the topic of the next part.

Part II

Physical Attacks and Countermeasures on ECC

Introduction

Side-Channel Analysis (SCA) is the cryptographic technique exploiting different leaks, such as the execution time, the power consumption, the electromagnetic emanation during the execution of a cryptographic algorithm embedded in a device. Kocher was the first to report a side-channel attack in 1996 [Koc96]. The attack exploits the variation of execution timing with different inputs. Boneh, DeMillo and Lipton introduced another kind of physical attack in 1997 [BDL97]. They suggest introducing a fault during the execution of the cryptographic algorithm. The secret key is then derived from the erroneous result. Since then, various fault attacks have emerged. On the other hand, many different countermeasures have been imagined and introduced year after year to defeat physical attacks. This part gives a survey on the physical attacks and countermeasures on ECC.

While defenders introduce new methods to thwart a class of attacks or an attack in particular, cryptanalysts propose new attacks to bypass previous countermeasures making them incomplete or even completely ineffective. We exhibit the attacks and countermeasures with a tree structure to represent this cat-and-mouse game.

Chapter 7 is devoted to the characterisation of the attacks and countermeasures. Considering the large number of attacks, the employment of precise terms to describe them is required. Also, most of the countermeasures have a negative effect on the performance; we explain how to quantify their cost.

Chapter 8 is the core of the thesis. It displays the different attacks and countermeasures. We propose new side-channel attacks, called Same-Values Analysis (SVA), based on the occurrence of same values within the same ECSM. Namely, these attacks are the horizontal SVA targeting the Unified Formulæ (Section 8.2.2.2), the vertical and horizontal SVA against the Side-Channel Atomicity (Sections 8.2.3.1 and 8.2.3.2), the vertical and horizontal classical SVA (Sections 8.13 and 8.14). Also, a new kind of fault attacks is proposed in Section 8.23. A fault is induced at the very end of the ECSM, during the conversion from projective to affine coordinates. In addition, we propose a very efficient countermeasure against the Refined Side-Channel Analysis in Section 8.11.1. We propose new elliptic curve formulæ, ensuring the security against the Refined Side-Channel Analysis.

Some fault attacks require several pairs of correct and faulted results. Intuitively, ECDSA seems naturally immune against such attacks because it is a probabilistic signature scheme. We show in Chapter 9 that is not true for several fault attacks.

Chapter 10 summarizes the attacks depending on the context. It is suitable for the task of selecting different countermeasures when implementing a secure embedded ECC application.

Finally, a synthesis on the attacks versus the countermeasures is given in Chapter 11. The efficiency of each countermeasure against the attacks is clearly displayed.

Chapter 7

Characterisation of Attacks and Countermeasures

Given the large number of physical attacks in ECC, the use of precise terms is essential to describe and classify them. In Section 7.1, we give the different categories of the attacks depending of the exploited leak. Section 7.2 gives a description of the context for the attacks. In the next chapter, experimental results are given to verify the practicability of some side-channel attacks. To perform the experiments, we use a test platform that is described in Section 7.3. Finally, we give in Section 7.4 the terms that will quantify the cost of each countermeasure.

7.1 Categories of the Attacks

The different attacks in embedded systems are generally classified into three main categories: **Side-Channel, Fault** and **Combined Attacks**. For each attack of the next chapter, we specify in which category it belongs to.

Side-Channel or **Passive Attacks** are those where the attacker observes the behaviour of the chip during a process without disturbing it. Since the behaviour depends on the manipulated data, the observations may reveal secret information such as the secret scalar. So far, the different passive attacks are:

- **Timing Attacks.** They exploit the interdependence between values of the inputs and the time needed to execute the cryptographic algorithm. The first reported side-channel attack of Kocher [Koc96] is in fact a timing attack.
- **Simple Side-Channel Analysis (SSCA).** The attacker observes the different patterns of the power consumption or the electromagnetic traces. Each step of the attack requires a single trace to conclude on some information of the secret.
- **Template Attacks.** They proceed in two phases [CRR02]. The first phase consists in building the templates. The attacker needs a fully controllable device in which she can choose private and public data, and acquire traces of the power consumption by varying the input data: this database makes up the templates. The second phase is the acquisition of the targeted device with the same known public data used for the templates. The trace is then compared with the templates to conclude which secret data are the more probably manipulated.

- **Vertical Side-Channel Analysis.** Several ECSMs are run with different input data. Each time, the power consumption or the electromagnetic radiation is acquired. A statistical analysis is performed on the different traces to deduce the manipulated values, and hence the secret scalar.

- **Horizontal Side-Channel Analysis.** A single trace is analysed to conclude on some information of the secret. The attacker uses statistical tools on segments of the trace. Since a single trace is available, the length of the random variables is really limited. Therefore, Horizontal SCA is more difficult to mount in practice compared with Vertical SCA in which the attacker has a potential access to unlimited traces. Nonetheless, recently, these attacks have been intensively studied because they are in fact very powerful since a single trace can reveal the whole secret data. These attacks make it possible to target some cryptographic protocols naturally immune to vertical SCA such as ECDSA. They also can bypass some very powerful countermeasures.

Fault or **Active Attacks** are those where the opponent disturbs the chip during the execution of the cryptographic algorithm by using a laser, by varying the supply voltage, by varying the external clock or other methods. The possibly erroneous result can reveal information of the targeted secret data. The first reported attack that used a fault to derive secret information was introduced by Boneh, DeMillo and Lipton [BDL97] to target RSA implementations used with the *Chinese Remainder Theorem* (CRT). The attack was renamed later *Bellcore attack* after the name of the company for which Boneh, DeMillo and Lipton were working. Since then, many methods to derive a secret key using a fault were introduced. In ECC, the different active attacks are generally classified as follows:

- **Safe-Error Attacks.** The attacker injects a fault on a specific area of the chip at a specific time during a process. The final result will be wrong *if and only if* some condition of the secret data is met. Otherwise, the result will be correct and the fault had no effect: it was safe.

- **Weak Curve Attacks.** A fault is induced on some parameters before the ECSM so that the ECSM is performed on another elliptic curve that is weaker than the original one. The new curve E' is weak in the sense that solving the ECDLP is easy. Generally, this means that the order of E' has no large factor.

- **Differential Fault Attacks (DFA).** Several ECSMs are run. Each time, a fault is induced during the execution of the cryptographic algorithm. The attacker collects the erroneous results. They are analysed and compared with each other or with the correct ones to deduce some information on the secret key. The term "Differential Fault Analysis" was first employed by Biham and Shamir to describe an attack against an implementation of the *Data Encryption Standard* (DES) [BS97]. The Bellcore attack on RSA is in fact a DFA.

This thesis focuses on the different methods to derive the secret scalar from possibly erroneous results rather than the technical aspect of injecting a fault. For a detailed description of the different methods to disturb the chip, the interested reader can refer to the survey given in [BCN+06].

Combined Attacks are those where the attacker combines two (or more) passive or active attacks at a time.

7.2 Attack Context

The context of an attack is essential for the designer trying to protect an embedded system. Depending on the protocol, the implementation or the architecture, an attack is not necessarily feasible. In this case, a protection is not necessary, and the performance can be increased because each countermeasure has a cost. The context is described by giving details on the following information:

- **Key Recovery.** A description of the key recovery procedure is given; it can be either recursive if the bits recovery process follows a certain order, or "independent bits" if the bits are recovered independently from each other. The key recovery process can also involve more computation such as the ECDLP to recover a small scalar, or the ECDLP on a weak curve.

- **Elliptic Curve Specificity.** Some attacks work only if the given elliptic curve has some properties; these properties are given,

- **Implementation Access.** The attacker needs some knowledge of the implementation (until a certain level of the ECC's hierarchy), or she needs an access to a device with exactly the same implementation.

- **Implementation Specificity.** An attack can either be adapted for many different implementations, or it targets a very specific implementation, a specific algorithm, a specific elliptic curve formula or a specific countermeasure.

- **Number of Executions Needed.** To succeed, an attack requires the run of one or several ECSMs (for example n, n being the bit-length of the scalar); when the attack is based on a statistical approach, we simply say "multiple" executions because the executions number depends on the standard deviation of the random variables.

- **Input Access.** Either the attacker needs to choose a base point having some properties or she simply needs the knowledge of it or the base point does not matter.

- **Output Access.** The attacker may need either some information about the output point like the knowledge of it or, more simply, the knowledge of its validity (correct or incorrect) or, even, no information at all about it.

- **Fault model.** The different faults are characterized as data randomization (a random fault is injected into a specific area), resetting data (a fault is injected to force a data to the zero value) or modifying opcode (a fault is injected to modify or skip some instructions). Also the area size of the fault is given.

7.3 Test Platform

Side-Channel attacks are generally experimentally validated. We implemented elliptic curves operations in the Side-channel Attack Standard Evaluation Board SASEBO-GII [SASEBO]. The hardware arithmetic module was implemented using algorithms described in Chapter 2 with a word size of 64 bits. All measures and experimental results given in the next chapter are performed with this test platform.

7.4 Quantifying the Cost of the Countermeasures

Each countermeasure has a non negligible time or memory cost on ECC applications. Indeed, a countermeasure can extend the ECSM by several iterations, it can increase the number of operations of an elliptic curve formula, or it can cost only a few modular multiplications. For the extra field operations, extra elliptic curve operations, or extra iterations of the ECSM, we use the notation of Sections 2.6, 3.2 and 4.3 respectively. A Random Number Generation (RNG), a Random Permutation Generation (RPG) or a Cyclic Redundancy Check (CRC) might be needed for the countermeasure. We notify it. The memory required to store the extra values also matters. For each countermeasure, we give the cost with the following notation:

- $\text{ECSM}_{l,n}$: execution time of an ECSM with a l-bit scalar and a n-bit modulus,
- $\text{ECADD}_n, \text{ECDBL}_n, \text{C-ECADD}_n$: execution time of an elliptic curve addition, doubling and conjugate addition with a modulus of size n,
- $\text{ADD}_n, \text{SQR}_n, \text{MUL}_n, \text{DIV}_n$: execution time of an addition/subtraction, a square, a multiplication and a division respectively, with n-bit integers,
- $\text{mADD}_n, \text{mSQR}_n, \text{mMUL}_n, \text{mINV}_n$: execution time of a modular addition/subtraction, a modular square, a modular multiplication and a modular inversion respectively, with n-bit integers,
- RNG_n: execution time of the generation of a random n-bit integer,
- RPG_m: execution time of the generation of a random permutation of m elements,
- CRC_n: execution time of a cyclic redundancy check of a n-bit integer,
- MEM_n: memory block to store a n-bit integer.

Chapter 8

Attacks and Countermeasures

This chapter gives a state-of-the-art in physical attacks and countermeasures on ECC. Existing attacks and countermeasures are described. Our new attacks: the Same-Values Analysis and the Fault Attack on Projective to Affine Conversion are detailed. Our countermeasure against the Refined Side-Channel Analysis is described as well.

Some countermeasures were introduced to counteract an attack in particular. In this case, the countermeasure will be a subsection of the latter. The same is done with an attack targeting a specific countermeasure. This structure tree is more suitable to expose the state-of-the-art. In addition, the titles of the attacks are in red, while the titles of the countermeasures are in blue.

For each attack, we specify if some countermeasures, prior in the state-of-the-art, thwart the attack. We also specify if previous countermeasures have been proven ineffective against the attack currently described. This analysis of interaction was either found in the paper introducing the attack or it is from our own researches.

The characteristics of the attacks, as described in the previous chapter, are given. Also, the cost of each countermeasure is specified.

Remark 8.1. Some passive attacks consist in analysing the power consumption trace during the execution of the cryptographic operation. These attacks can be adapted using the electromagnetic radiation trace. We use a single notation for better clarity.

In this chapter, unless otherwise specified, we consider the following parameters:

- $E\colon y^2 = x^3 + ax + b$ is the given elliptic curve defined over $\mathbb{F}_p$ with p a n-bit prime integer,
- k is the secret scalar that the attacker tries to recover.

8.1 Classical Timing Attack [Koc96]

The first timing attack, introduced by Kocher, targets RSA implementations. It takes advantage of the non-constant execution timing of the modular multiplication such as Algorithm 8 with the conditional reduction step. The attack was improved and simplified by Dhem, Koeune, Leroux, Mestré, Quisquater and Willems [DKL+98]. It is described below, adapted to ECC.

The attack is recursive. Suppose that the attacker already knows the $n-i-1$ leftmost bits of the fixed scalar $k = (k_{n-1}, \ldots, k_0)_2$ and tries to recover k_i ([1]).

The attacker collects the execution time of different ECSMs with different base points. She simulates the computation with exactly the same implementation of the targeted chip, by making an assumption on k_i (e.g. $k_i = 0$).

She separates the different timings in two sets S_1 and S_2. If a final reduction is needed at a specific time of the algorithm (this can be for example the first multiplication of the elliptic curve operation), the timing is put in S_2. The timing is put in S_1 otherwise (no reduction). Let T_1 and T_2 be the average timings of S_1 and S_2 respectively. If $T_2 - T_1 \approx \varepsilon$, ε being the average time of the final reduction, then the hypothesis on the secret was right.

Remark 8.2. In [DKL+98], they improve the attack by making the two possible assumptions: $k_i = 0$ and $k_i = 1$. The sets T_1 (no reduction) and T_2 (reduction) are constructed for both assumptions. They are denoted by $T_1^{(0)}, T_2^{(0)}$ and $T_1^{(1)}, T_2^{(1)}$ for $k_i = 0$ and $k_i = 1$, respectively. The method consists in comparing $T_2^{(0)} - T_1^{(0)}$ and $T_2^{(1)} - T_1^{(1)}$. The larger number reveals the good hypothesis.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: full knowledge of all algorithms,
- **Implementation Specificity**: deterministic and non-constant time execution of modular multiplications,
- **Number of Executions Needed**: multiple,
- **Input Access**: known and varying,
- **Output**: unnecessary.

8.1.1 Constant Time of Field Operations

A effective solution to prevent the timing attack described above is to perform all field operations in constant time, whatever the input values. This can be done by adding dummy operations if necessary as explained in Sections 2.3 and 2.4. To ensure a full protection, the designer should verify that the the field operations are performed with the same number of cycles whatever the input values.

With random values, the probability of subtraction for a modular addition is 1/2, and the probability of addition for a modular subtraction is 1/2. The probability of a final reduction is 3/16 for a Montgomery multiplication and 1/4 for a Montgomery squaring [SST04]. One extra memory block is required to store the result of dummy operations.

Cost: 1/2 ADD_n per mADD_n, 13/16 ADD_n per mMUL_n, 3/4 ADD_n per mSQR_n, 1 MEM_n

[1] This presumes this is a Left-to-Right ECSM. Of course, the attack works backwards on Right-to-Left ECSMs.

8.2 Simple Side-Channel Analysis [Cor99]

Coron was the first to report a SSCA on ECC. This attack targets unregular ECSMs such as the Left-to-Right Double-and-Add method (Algorithm 14).

At each iteration, an addition of points is performed only if the current bit of the scalar is 1. If the attacker is able to distinguish the power consumption of a doubling from the one of an addition of points, as in Figure 8.1, the bits of the scalar are easily recovered. Contrary to RSA, where squares and multiplications can be executed by the same code, this is not the case for ECC.

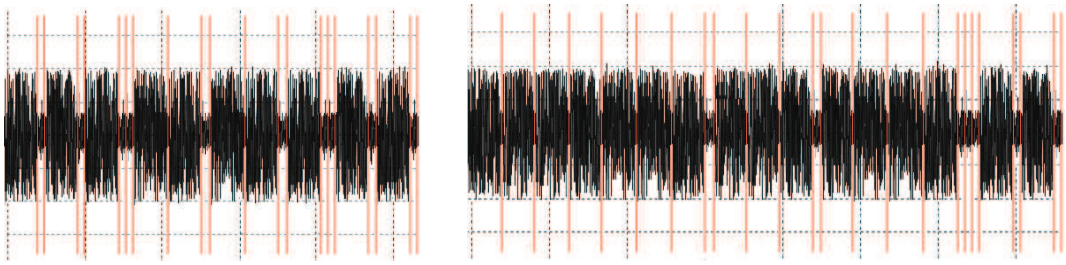


Figure 8.1: Power consumption trace of a doubling (left curve) and an addition (right curve). Field operations are delimited by vertical lines.

A single trace of an ECSM is enough to recover the scalar.

Attack Context:

- **Key recovery**: recursive (the iteration number has to be known),
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: unregular ECSM, different formulæ for addition and doubling operations,
- **Number of Executions Needed**: 1,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

8.2.1 Regular ECSM [Cor99, §3.1]

A countermeasure against SSCA consists in regularizing the ECSM, *i.e.* perform the same elliptic curve operations at each iteration of the ECSM whatever the value of the scalar.

Introducing dummy operations can make an ECSM regular such as the Left-to-Right Double-and-Add always (Algorithm 19). The Montgomery Ladder (Algorithms 21 and 22) is regular as well without any dummy operation. Regular ECSMs were previously given in Section 4.2.

8.2.1.1 C Safe-Error [YKLM01]

C Safe-Error attacks, introduced by Yen, Kim, Lim and Moon, target implementations with dummy operations such as the Left-to-Right Double-and-Add always method (Algorithm 19).

The attacker injects a fault during the computation of the addition of points at iteration i. If the addition is dummy, which is the case if $k_i = 0$, the fault had no effect and the result is correct.

This attack targets one bit at a time per ECSM.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the field arithmetic module, knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: implementation with dummy operations depending on the current bit,
- **Number of Executions Needed**: n,
- **Input Access**: unnecessary,
- **Output Access**: knowledge of the validity,
- **Fault Model**: any on the field arithmetic module.

8.2.2 Unified Formulæ [BJ02]

This countermeasure was introduced by Brier and Joye to prevent the SSCA. Elliptic curve operations are reviewed so that the operations for computing a doubling and an addition are the same.

The following formula can be used for both addition ($Q \neq \pm P$) and doubling ($Q = P$). They presented the formula in projective coordinates because it is more efficient than the Jacobian coordinates.

Algorithm 31 unified ECADD in projective coordinates [BJ02]

Input: $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$ in projective coordinates such that $Y_1 Z_2 \neq -Y_2 Z_1$, elliptic curve parameter a
Output: $P + Q$

$U_1 \leftarrow X_1 Z_2$; $U_2 \leftarrow X_2 Z_1$; $S_1 \leftarrow Y_1 Z_2$; $S_2 \leftarrow Y_2 Z_1$
$Z \leftarrow Z_1 Z_2$; $T \leftarrow U_1 + U_2$; $M \leftarrow S_1 + S_2$
$R \leftarrow T^2 - U_1 U_2 + aZ^2$; $F \leftarrow ZM$
$L \leftarrow MF$
$G \leftarrow TL$
$W \leftarrow R^2 - G$
$X_3 \leftarrow 2FW$
$Y_3 \leftarrow R(G - 2W) - L^2$
$Z_3 \leftarrow 2F^3$
return (X_3, Y_3, Z_3)

Cost: (unified ECADD): 13 mMUL_n, 5 mSQR_n, 9 mADD_n, 6 MEM_n

This prevents the classical SSCA since doubling and addition operations cannot be distinguished with the power consumption trace as in Figure 8.1.

This countermeasure must be carefully implemented. The attacker should not be able to distinguish the transition of an iteration to the next from the transition of the two elliptic curve operations within the same iteration. The atomicity principle should be applied [CCJ04].

An addition of points requires 13 mMUL_n, 5 mSQR_n, 9 mADD_n and 6 temporary registers.

Cost:

- 9 $\mathrm{mMUL}_n - 1$ $\mathrm{mSQR}_n - 2$ mADD_n over ECDBL_n,
- 1 $\mathrm{mMUL}_n + 1$ $\mathrm{mSQR}_n + 2$ mADD_n over ECADD_n.

8.2.2.1 SSCA on Unified Formulæ [Wal04]

Walter showed that a simple SCA can reveal whether a final subtraction is needed at the end of the Montgomery multiplication, as shown in Figure 8.2. By analysing all Montgomery multiplications performed during the execution of Algorithm 31, the attacker is able to tell if the points are the same (doubling) or not (addition).

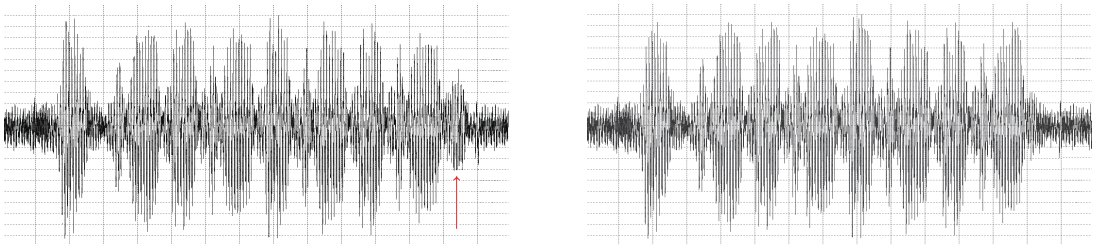


Figure 8.2: Power consumption of a multiplication with a final subtraction (left curve) and without a final subtraction step (right curve). The vertical arrow indicates the additional subtraction.

Stebila and Thériault improved the attack [ST06]. In addition to the conditional subtraction at the end of the Montgomery multiplication, they use the conditional subtraction (resp. addition) at the end of the modular addition (resp. subtraction).

A single trace may be enough to recover the whole bits of the scalar. In addition, the knowledge of the input point is not required [Wal04, ST06]. Obviously, the Constant Time of Field Operations countermeasure, as described in Section 8.1.1, thwarts the attack[2].

Attack Context:

- **Key recovery**: recursive (the iteration number has to be known),
- **Elliptic Curve Specificity**: unregular ECSM,

[2]Of course, only if the dummy operations are performed in the same manner as if they were real. Also, the conditional step and throwing away the temporary wrong result should not be distinguished with side-channel analysis.

- **Implementation Access**: full knowledge of all algorithms,
- **Implementation Specificity**: deterministic and non-constant time execution of modular multiplications, unified formulæ,
- **Number of Executions Needed**: 1,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

8.2.2.2 Horizontal Same Values Analysis on Unified Formulæ

We introduce a new attack against the Unified Formulæ countermeasure. A single trace is analysed: it is a horizontal attack.

Algorithm 31 is used so that the attacker cannot distinguish whether a doubling or an addition is being performed. However, if the input points of unified ECADD are the same, the values U_1, U_2 are computed with the same input values: $X_1 = X_2$ and $Z_1 = Z_2$. The same goes for S_1, S_2: $Y_1 = Y_2$ and $Z_1 = Z_2$.

The trace segments during the computation of U_1, U_2 are compared as illustrated in Figure 8.3.

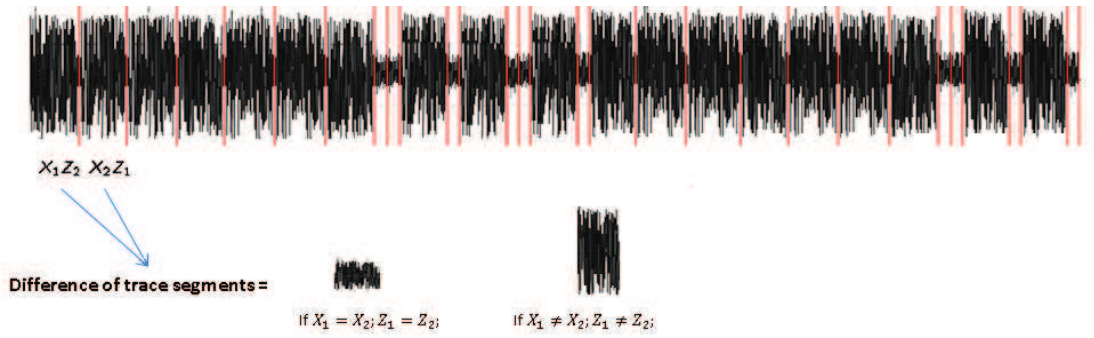


Figure 8.3: Comparison of the power consumption during the computation of unified ECADD.

If the inputs of the two multiplications are equal, the difference of traces corresponds of the difference of the noise only. Consequently, if the noise is low, the difference of traces will be near zero in that case. If the noise is high, a more sophisticated tool than a simple difference can be used, such as the Euclidean distance or the correlation with the points of interest of the traces. The trace segments during the computation of U_1, U_2 can be seen as random variables X, Y respectively. The construction of such random variables are illustrated in Figure 8.4.

The comparison can be done at each iteration, so the scalar can be recovered from a single trace.

Attack Context:

- **Key recovery**: recursive (the iteration number has to be known),
- **Elliptic Curve Specificity**: none,

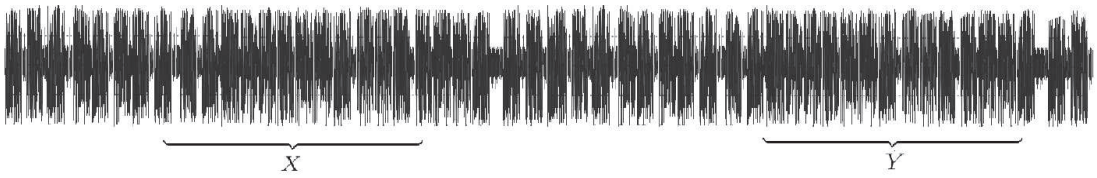


Figure 8.4: Construction of random variables for a Horizontal SCA without a leakage model during the execution of an ECSM.

- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: unified formula,
- **Number of Executions Needed**: 1,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

8.2.3 Side-Channel Atomicity [CCJ04]

The concept of *Side-Channel Atomicity* was introduced by Chevallier-Mames, Ciet and Joye. The elliptic curve formulæ are rewritten with sequences of identical *atomic patterns*. In [CCJ04], an atomic pattern is the sequence of the following (possibly fake) operations:

1. modular multiplication or square
2. modular addition
3. modular opposite
4. modular addition

A point doubling requires 10 of these atomic patterns, while an addition requires 16.

This countermeasure has been improved by Giraud and Verneuil [GV10]. A point doubling in modified coordinates requires 2 mSQR_n, 6 mMUL_n, 10 mADD_n and 7 temporary registers[3]. An addition of points requires the double amount of operations: 4 mSQR_n, 12 mMUL_n, 20 mADD_n and 7 temporary registers[3]. The formulæ are suitable for the Right-to-Left sliding window width-v NAF scalar multiplication (Algorithm 17). The elliptic curve operations are illustrated in Figure 8.5.

This implementation is not vulnerable anymore to SSCA since the attacker cannot distinguish between the operations performed. She only sees a sequence of identical atomic patterns without knowing if they correspond to a doubling or an addition (see Figure 8.6).

Fake operations are introduced so that the different elliptic curve operations might be written with the same atomic patterns. Therefore, Safe-Error attacks can be applied to distinguish which operation is currently performed.

[3] 6 temporary registers are needed in [GV10], we added one extra temporary registers for the dummy operations.

	ECADD - part 1 ($\mathcal{A}1$)	ECADD - part 2 ($\mathcal{A}2$)	modECDBL ($\mathcal{D}$)
1.	$T_1 \leftarrow Z_2^2$	$T_1 \leftarrow T_6^2$	$T_1 \leftarrow X_1^2$
2.	$\star$	$\star$	$T_2 \leftarrow Y_1 + Y_1$
3.	$T_2 \leftarrow Y_1 \times Z_2$	$T_4 \leftarrow T_5 \times T_1$	$Z_3 \leftarrow T_2 \times Z_1$
4.	$\star$	$\star$	$T_4 \leftarrow T_1 + T_1$
5.	$T_5 \leftarrow Y_2 \times Z_1$	$T_5 \leftarrow T_1 \times T_6$	$T_3 \leftarrow T_2 \times Y_1$
6.	$\star$	$\star$	$T_6 \leftarrow T_3 + T_3$
7.	$T_3 \leftarrow T_1 \times T_2$	$T_1 \leftarrow Z_1 \times T_6$	$T_2 \leftarrow T_6 \times T_3$
8.	$\star$	$\star$	$T_1 \leftarrow T_4 + T_1$
9.	$\star$	$\star$	$T_1 \leftarrow T_1 + W_1$
10.	$T_4 \leftarrow Z_1^2$	$T_6 \leftarrow T_2^2$	$T_3 \leftarrow T_1^2$
11.	$T_5 \leftarrow T_5 \times T_4$	$Z_3 \leftarrow T_1 \times Z_2$	$T_4 \leftarrow T_6 \times X_1$
12.	$\star$	$T_1 \leftarrow T_4 + T_4$	$T_5 \leftarrow W_1 + W_1$
13.	$T_2 \leftarrow T_2 - T_3$	$T_6 \leftarrow T_6 - T_1$	$T_3 \leftarrow T_3 - T_4$
14.	$T_5 \leftarrow T_1 \times X_1$	$T_1 \leftarrow T_5 \times T_3$	$W_3 \leftarrow T_2 \times T_5$
15.	$\star$	$X_3 \leftarrow T_6 - T_5$	$X_3 \leftarrow T_3 - T_4$
16.	$\star$	$T_4 \leftarrow T_4 - X_3$	$T_6 \leftarrow T_4 - X_3$
17.	$T_6 \leftarrow X_2 \times T_4$	$T_3 \leftarrow T_4 \times T_2$	$T_4 \leftarrow T_6 \times T_1$
18.	$T_6 \leftarrow T_6 - T_5$	$Y_3 \leftarrow T_3 - T_1$	$Y_3 \leftarrow T_4 - T_2$

Figure 8.5: ECADD and modECDBL operations written with the same atomic pattern ($\star$ represents a dummy operation). Each column is an atomic pattern.

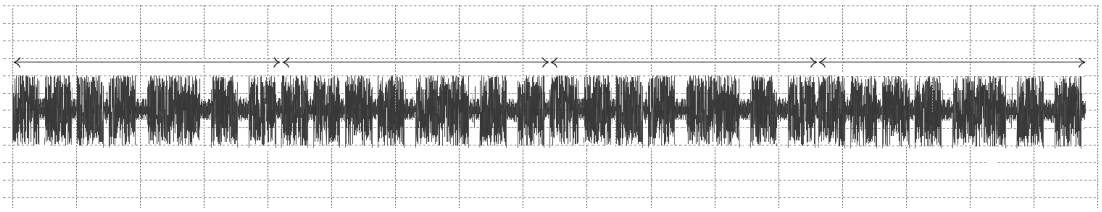


Figure 8.6: Power trace observed during a few iterations of the ECSM using the patterns of Figure 8.5

Cost:

- modECDBL$_n$ with atomic patterns costs 2 mMUL$_n$ − 2 mSQR$_n$ − 2 mADD$_n$ + 4 MEM$_n$ over the classical modECDBL$_n$ of Section 3.1.1 (which is an improvement if mMUL$_n$ = mSQR$_n$),
- ECADD$_n$ with atomic patterns costs 13 mADD$_n$ + 4 MEM$_n$ over the classical ECADD$_n$ of Section 3.1.1.

8.2.3.1 Same Values Analysis against the Atomicity Countermeasure

We present our Vertical Side-Channel Attack targeting the Side-Channel Atomicity Countermeasure. We use the notation of the patterns $\mathcal{A}1, \mathcal{A}2, \mathcal{D}$ as described in the previous section. We illustrate our attack on the Right-to-Left sliding window width-v NAF scalar multiplication (Algorithm 17) in which the Side-Channel Atomicity was described in [CCJ04, GV10].

If we are able to distinguish between a doubling ($\mathcal{D}$) and an addition ($\mathcal{A}1;\mathcal{A}2$), we can deduce if the current digit of the scalar is zero. The attack is recursive. For a better clarity, we will see how to find the first digit k_0 of the v-NAF representation of k. The next digits are recovered in the same way.

The core idea of the attack is to identify which operation is performed by analysing the possible repetitions of variables in the patterns.

Possibilities of the patterns. Considering Algorithm 17, the possible operations of the three first atomic patterns are:

1. $\mathcal{A}1;\mathcal{A}2;\mathcal{D}$. In this case, $k_0 \neq 0$.
2. $\mathcal{D};\mathcal{A}1;\mathcal{A}2$. In this case, $k_0 = 0$.
3. $\mathcal{D};\mathcal{D};\mathcal{A}1$. In this case, $k_0 = 0$.
4. $\mathcal{D};\mathcal{D};\mathcal{D}$. In this case, $k_0 = 0$.

We want to assert if the first three patterns correspond to $\mathcal{A}1;\mathcal{A}2;\mathcal{D}$ ($k_0 \neq 0$).

Same values in the different patterns. With Figure 8.5 and the different possibilities of the three first patterns, we label the modular multiplications with a common operand *only* if the operations are $\mathcal{A}1;\mathcal{A}2;\mathcal{D}$; we deliberately omit the multiplications sharing a common operand if they possibly occur in another sequence of patterns.

The common operands are illustrated in Figure 8.7. They are denoted with boxes with the same index. For example, the square at line 1 of the 1^{st} pattern and the multiplication at line 3 of the 1^{st} pattern share a common operand (Z_2) only if the sequence is $\mathcal{A}1;\mathcal{A}2;\mathcal{D}$. Note that the multiplication at line 17 of the 1^{st} pattern and the multiplication at line 11 of the 3^{rd} pattern share a common operand (X_2 and X_1) only if $\mathcal{A}1;\mathcal{A}2;\mathcal{D}$ is performed. The same holds for Z_2 in $\mathcal{A}1;\mathcal{A}2$ and Z_1 in $\mathcal{D}$. Indeed, the point (X_2, Y_2, Z_2) of $\mathcal{A}1;\mathcal{A}2$ and the point (X_1, Y_1, Z_1) of $\mathcal{D}$ both correspond to the point R or $-R$ in Algorithm 17.

The total number of pairs of multiplications or squares sharing a common operand is sixteen in the sequence $\mathcal{A}1;\mathcal{A}2;\mathcal{D}$.

Detecting the same values. The attacker arbitrarily chooses a pair within the sixteen. Several ECSMs are run with the same scalar k. The base point does not matter.

We want to detect if, for each ECSM, the two multiplications of the selected pair share a common operand. The method introduced by Schramm et al. [SWP03] to attack an implementation of the DES can be used. It was later improved in [CFG+11] to attack a protected implementation of the *Advance Encryption Standard* (AES). The principle of the method is as follows. Within each trace, the two points where the same values are possibly manipulated are saved for constructing two random variables, as illustrated in Figure 8.8. The correlation or another statistical tool is performed to reveal if the same values are indeed manipulated.

Remark 8.3. The pair of multiplications is selected such that the common operands are in the same operand input (left or right). This makes it possible to perform a collision analysis without any synchronization procedure. The attack is still possible if it is not the case, but a strong study on the field arithmetic module is required for a synchronization because the sensitive data is not manipulated at the same time within the field operation.

	ECADD - part 1 ($\mathcal{A}1$)	ECADD - part 2 ($\mathcal{A}2$)	modECDBL ($\mathcal{D}$)
1.	$T_1 \leftarrow \boxed{Z_2}^2_{1,2,14}$	$T_1 \leftarrow \boxed{T_6}^2_{9,10}$	$T_1 \leftarrow \boxed{X_1}^2_{12}$
2.	$\star$	$\star$	$T_2 \leftarrow Y_1 + Y_1$
3.	$T_2 \leftarrow Y_1 \times \boxed{Z_2}_{1,3,15}$	$T_4 \leftarrow T_5 \times T_1$	$Z_3 \leftarrow T_2 \times \boxed{Z_1}_{14,15,16}$
4.	$\star$	$\star$	$T_4 \leftarrow T_1 + T_1$
5.	$T_5 \leftarrow Y_2 \times \boxed{Z_1}_{4,5}$	$T_5 \leftarrow T_1 \times \boxed{T_6}_{9,11}$	$T_3 \leftarrow T_2 \times Y_1$
6.	$\star$	$\star$	$T_6 \leftarrow T_3 + T_3$
7.	$T_3 \leftarrow \boxed{T_1}_7 \times T_2$	$T_1 \leftarrow \boxed{Z_1}_{5,6} \times \boxed{T_6}_{10,11}$	$T_2 \leftarrow T_6 \times T_3$
8.	$\star$	$\star$	$T_1 \leftarrow T_4 + T_1$
9.	$\star$	$\star$	$T_1 \leftarrow T_1 + W_1$
10.	$T_4 \leftarrow \boxed{Z_1}^2_{4,6}$	$T_6 \leftarrow T_2^2$	$T_3 \leftarrow T_1^2$
11.	$T_5 \leftarrow T_5 \times \boxed{T_4}_8$	$Z_3 \leftarrow T_1 \times \boxed{Z_2}_{2,3,16}$	$T_4 \leftarrow T_6 \times \boxed{X_1}_{13}$
12.	$\star$	$T_1 \leftarrow T_4 + T_4$	$T_5 \leftarrow W_1 + W_1$
13.	$T_2 \leftarrow T_2 - T_3$	$T_6 \leftarrow T_6 - T_1$	$T_3 \leftarrow T_3 - T_4$
14.	$T_5 \leftarrow \boxed{T_1}_7 \times X_1$	$T_1 \leftarrow T_5 \times T_3$	$W_3 \leftarrow T_2 \times T_5$
15.	$\star$	$X_3 \leftarrow T_6 - T_5$	$X_3 \leftarrow T_3 - T_4$
16.	$\star$	$T_4 \leftarrow T_4 - X_3$	$T_6 \leftarrow T_4 - X_3$
17.	$T_6 \leftarrow \boxed{X_2}_{12,13} \times \boxed{T_4}_8$	$T_3 \leftarrow T_4 \times T_2$	$T_4 \leftarrow T_6 \times T_1$
18.	$T_6 \leftarrow T_6 - T_5$	$Y_3 \leftarrow T_3 - T_1$	$Y_2 \leftarrow T_4 - T_2$

Figure 8.7: Common operands in the atomic patterns

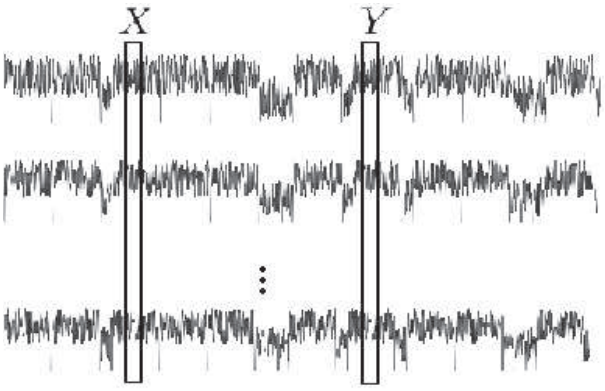


Figure 8.8: Construction of random variables for a detection of same values

Experimental Results. We conducted an experiment with the test platform described in Section 7.3. We measured the power consumption of 400 pairs of modular multiplications of 256-bit integers. For each pair, the two modular multiplications share a common operand. We then computed the correlation curve (X and Y illustrated in Figure 8.8 are slid together along the traces). The same was done with 400 pairs of modular multiplications with random operands. The correlation curves are given in Figure 8.9. Four peaks can be seen when the

modular multiplications share a common operand. It corresponds to the four words of the common operand.

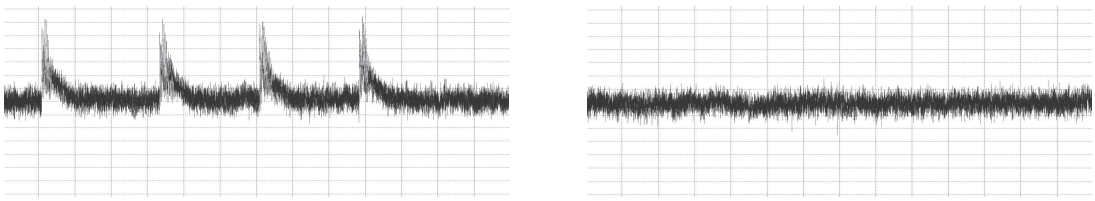


Figure 8.9: Correlation curves results for 400 pairs of modular multiplications sharing a common operand (left curve) and random operands (right curve) of 256-bit integers

If a significant peak can be seen on the correlation curve (left curve of Figure 8.9), the operations of the three first patterns are $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$. The attacker concludes that $k_0 \neq 0$. She starts again with the next three patterns to target the digit k_1. Conversely, if no peak is detected (right curve of Figure 8.9), the first pattern corresponds to $\mathcal{D}$, and the attackers concludes that $k_0 = 0$. The attacker starts again with the two last patterns of the three, added together with the fourth of the ECSM to target k_1.

The fifteen other pairs can be used to increase the length of the random variables and therefore decrease the number of traces required.

Attack Context:

- **Key recovery**: recursive (the iteration number has to be known),
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: unregular ECSM, usage of the Side-Channel Atomicity countermeasure,
- **Number of Executions Needed**: multiple,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

8.2.3.2 Horizontal SVA on the Atomicity Countermeasure

The previous attack was a Vertical Side-Channel Attack. The attack is no longer applicable if the scalar is not fixed or randomized.

We extend the previous attack and propose a Horizontal Side-Channel Attack. Like the previous attack, the attacker tries to detect if the three current patterns are $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$ using the possibly same values occurring in the patterns. Our attack is based on the Big Mac principle.

Big Mac attack. The Big Mac attack was first introduced by Walter to target RSA implementations [Wal01]. The attack is based on a method to detect if two multiplications share a common operand by comparing their power trace. This method is described here because we extend this method for our attack. The Big Mac attack will be fully described later in Section 8.7.

Walter pointed out that two multiplications with a common operand have similarities as for the power consumption. Denote by T_1, T_2 the traces during the computation of respectively two modular multiplications $A \times B \bmod P$, $C \times D \bmod P$, with $A \neq C$ (or Montgomery multiplications).

Denote by m the words number of the integers. The sample points of the trace T_1, in which each $b_j, j \in [0, m[$ is manipulated, are averaged into one single value $s_1^{(j)}$. If Algorithm 8 is used, this corresponds to the lines 8 to 10. The computation of $s_1^{(0)}$ is illustrated in Figure 8.10.

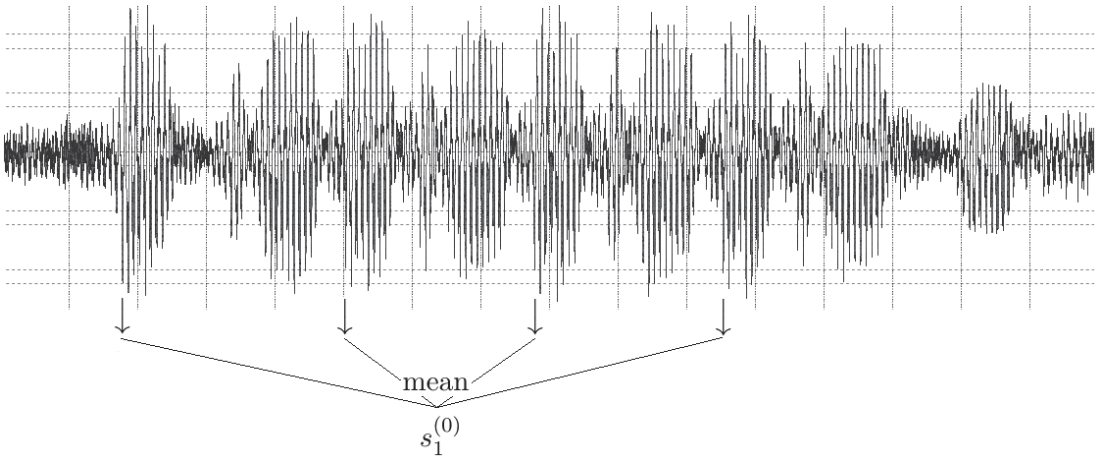


Figure 8.10: Illustration of the computation of $s_1^{(0)}$ with a modular multiplication of integers of four words (256-bit integers in a 64-bit architecture)

The average permits to reduce the noise corresponding to the manipulation of the a_i for $i \in [0, m[$ and other activities of the device [Wal01]. Let $S_1 = s_1^{(0)} || \ldots || s_1^{(m-1)}$ be the concatenation of the $s_1^{(j)}$. The same is done with T_2 to obtain S_2.

If $B = D$, the Euclidean distance between S_1 and S_2 is small. If $B \neq D$, the distance is high. The trace segments T_1, T_2 can be seen as random variables constructed from a single trace of an exponentiation, as illustrated in Figure 8.4.

This method works with a single pair of multiplications on RSA because the number of words is large compared to ECC[4]. The success of the attack depends on the size of the manipulated integers: the longer the used integers are, the higher the success rate is [Wal01, BJPW13a].

Big Mac CoCo. In [CFG+12], Clavier et al. propose to use the Pearson coefficient between S_1, S_2 instead of the Euclidean distance. They called the method Big Mac CoCo (CoCo for *Collision-Correlation*). They compare the two methods with simulation results on RSA and the correlation is better by far [CFG+12].

Later, Bauer, Prouff, Jaulmes and Wild gave simulation results of the Big Mac CoCo on elliptic curves [BJPW13b]. They target the Side-Channel Atomicity. Indeed, they notice that there are common operands regarding the side-channel atomicity formulæ. For instance, to distinguish an addition from a doubling, they suggest to compare the first multiplication (line 1) and the second multiplication (line 3) of Figure 8.5. If it is a doubling, the two multiplications

[4]For a 128 bits security, ECC must use 256-bit integers length, while RSA must use 3072-bit integers.

share a common operand. They give the success rate on simulation results using a correlation which was high enough even for a 32 bits architecture.

We experimentally tried both the Big Mac and the Big Mac CoCo on real measurements on a 64 bits architecture and we failed. In the following, we present a significant improvement of the attack of [BJPW13b]. We also present experimental results of our attack.

Assembling the pieces of the puzzle. In [BJPW13b], they compare only two multiplications in two different patterns. Our contribution is to compare many pairs of multiplications by analysing a sequence of several patterns depending of a bit of the scalar. Namely, we want to assert if the three first patterns correspond to $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$. In this case, the first bit is non-zero.

Compared with the classical Big Mac attack on RSA, the low number of words is compensated by the large number of modular multiplications we compare. We can compare sixteen pairs (see Figure 8.7) instead of one, thanks to the atomicity countermeasure.

First, we split the trace of the three first patterns; we separate the field operations. We denote by $s(\cdot)$ the method for constructing S_1 or S_2 as previously described for the Big Mac attack.

We then construct two sets U_1, U_2 as follows. U_1, U_2 are first set empty. We perform $s(\cdot)$ for the power traces of the multiplications that might share a common operand. One element of each pair is put in U_1, the other is put in U_2. The construction of U_1, U_2 is illustrated in Figure 8.11 for the first three pairs possibly sharing the same operand Z_2.

The Euclidean distance between U_1 and U_2 is small if each pair share a common operand. In this case the three patterns observed are actually $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$, and the attacker concludes that $k_0 \neq 0$. She then iterates the method with the next three patterns to target the digit k_1. Conversely, The Euclidean distance between U_1 and U_2 is large if no multiplication among all multiplications shares a common operand. In this case, the three patterns observed are not $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$, and the attacker concludes that $k_0 = 0$. She starts again with the two last patterns of the three, added together with the fourth pattern of the ECSM to target k_1.

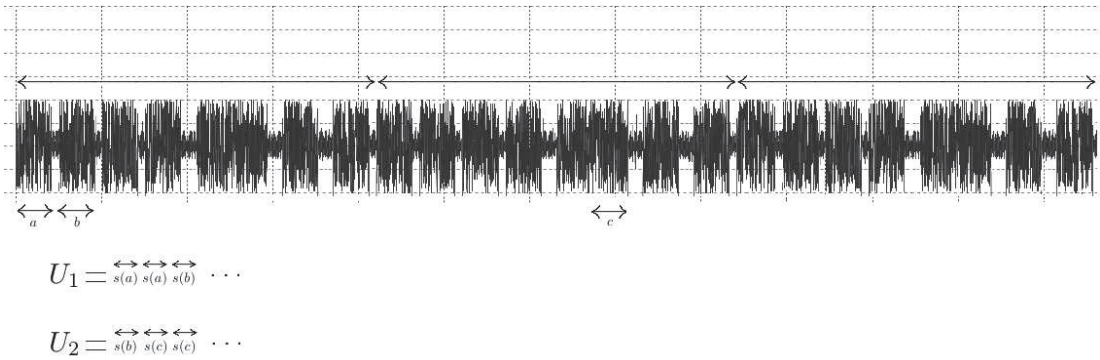


Figure 8.11: Assembling the pieces of the puzzle of three atomic patterns

Experimental Results - Euclidean Distance. We mounted the attack with 384-bit integers (six words with our 64-bit architecture) with our test platform described in Section 7.3.

The first step of the attack is the characterisation of the arithmetic module. We constructed

U_1, U_2 as previously described with fourteen pairs of multiplications sharing a common operand[5] 100 times. The average Euclidean distance was 2.165. The same was done with fourteen pairs of multiplication with random operands. The average Euclidean distance was 3.198. We established that a distance lower than the mean 2.682 correspond to $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$.

We then assembled the pieces of the puzzle as previously described with a trace of $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$ 50 times. Only one distance was larger than 2.682. We conclude that the attacker can detect $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$ with a success of 98%. The same was done with $\mathcal{D}; \mathcal{D}; \mathcal{A}1$ 50 times. Only two distances were smaller than 2.682. We conclude that the attacker wrongly detects a patterns triplet as $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$ with probability 4%.

We performed the experiment with 256-bit integers (four words) as well. We obtained a probability of 96% to correctly detect $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$, and a probability of 16% that $\mathcal{D}; \mathcal{D}; \mathcal{A}1$ was detected as $\mathcal{A}1; \mathcal{A}2; \mathcal{D}$, which is still acceptable to perform the attack.

We strongly believe that the success probability is higher on a 32-bit architecture because of the larger number of words.

Experimental Results - Big Mac CoCo. We also tried using the Pearson correlation as in [CFG+12, BJPW13b]. Surprisingly, the coefficient was high (around 0.9) each time, even if the guess was incorrect (i.e. even if there are no common operand for all multiplications).

The reason is that there are similarities in long integer multiplications even if the values are different such as the word index. Our experiment shows that in certain cases, the Euclidean Distance is better than the correlation. It also shows how difficult it is to characterise the leakage of an implementation with simulated leaks. We think that, at this stage of research, the best method is to experimentally try different methods and pick the best one.

Unlike the vertical version (Section 8.2.3.1), only a single trace is analysed. The secret scalar can thus be recovered with a single execution of the ECSM.

Attack Context:

- **Key recovery**: recursive (the iteration number has to be known),
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: full knowledge of all algorithms, knowledge of the architecture of the modular arithmetic operator,
- **Implementation Specificity**: unregular ECSM, usage of the Side-Channel Atomicity countermeasure, word-wise method of the modular multiplication,
- **Number of Executions Needed**: 1,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

Note that we introduced side-channel attacks against the Unified Formulæ and the Side-Channel Atomicity. Both countermeasures were proposed to prevent the SSCA. Furthermore, our attacks are very powerful since they require only a single trace and the knowledge of the input is not necessary. We strongly suggest to use regular ECSMs, the only remaining protection against the SSCA that has not been directly targeted by powerful attacks.

[5] We use fourteen pairs instead of sixteen as shown in Figure 8.7 because we avoid the pairs where the possibly same operand is not in the same side: boxes 5 and 13.

8.3 Correlation Side-Channel Analysis [Cor99, §3.2]

Coron was the first to report a Vertical Side-Channel Analysis on ECC. This attack is called Correlation Side-Channel Analysis (CSCA).

The attack is recursive. Suppose that the attacker already knows the $n-i-1$ leftmost bits of the fixed scalar $k = (k_{n-1}, \ldots, k_0)_2$ and tries to recover k_i ([6]).

The attacker collects N power consumption traces during the iteration i of the ECSMs with different base points. Each time, she tries to guess the values that are manipulated, by making an assumption on k_i. The guessed values are denoted by a_j with $j \in [1, N]$. The a_j are indeed manipulated only if her guess is correct.

She constructs the random variable X with the samples of the traces where the possible values a_j are used. Besides, she constructs the random variable Y with these values a_j and a specific leakage model $\mathbf{m}$ (e.g. the Hamming weight or the Hamming distance). The correlation between X and Y is high if the hypothesis on the secret data is correct. Figure 8.12 illustrates the construction of X and Y.

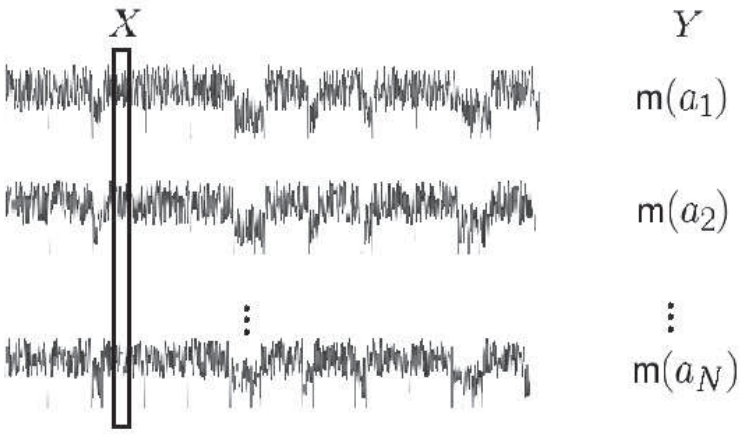


Figure 8.12: Construction of random variables for a CSCA

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM, the elliptic curve formulæ, the representation of integers and the word size of manipulated integers,
- **Implementation Specificity**: none,
- **Number of Executions Needed**: multiple,
- **Input Access**: known and varying,
- **Output Access**: unnecessary.

[6] This presumes this is a Left-to-Right ECSM. Of course, the attack works backwards on Right-to-Left ECSMs.

In the same paper introducing the CSCA on ECC, Coron exposes three very different methods to thwart his own attack: the Group Scalar Randomization (Section 8.3.1), the Point Blinding (Section 8.3.5) and the Random Projective Coordinates (Section 8.3.6). These methods were intensively studied, attacked or improved during the past years.

8.3.1 Group Scalar Randomization [Cor99, §5.1]

Coron suggests to randomize the scalar using the group structure of the elliptic curve. The scalar is randomized as $k' = k + r\#E$, with a random r of small size (32 bits seems a good trade of between security and efficiency). Obviously, the result of the ECSM is the same: $[k + r\#E]P = [k]P + [r]([\#E]P) = [k]P$ for any point on the curve.

The countermeasure prevents the CSCA because the attack works only on a fixed scalar. It prevents the classical Timing attack (see Section 8.1) for the same reason. The countermeasure thwarts also the C Safe-Error attack (see Section 8.2.1.1). Indeed, this attack can target a single bit at a time. Recovering a single bit of k' is useless.

k and $\#E$ have generally the same size n. So k' is approximatively a $(n + 32)$-bit integer. The ECSM is longer by 32 iterations.

Cost: 1 mMUL_n, 1 ADD_{n+32}, 1 MEM_{32}, 1 RNG_{32}, 1 $\text{ECSM}_{32,n}$

8.3.1.1 Carry Leakage Attack [FRVD08]

Fouque, Réal, Valette and Drissi introduced the Vertical Attack called the Carry Leakage Attack to target the Group Scalar Randomization.

This attack consists in analysing the carry propagation of the addition performed for the scalar randomization. The probability to have a carry only depends on the most significant bits of each word. The power consumption of several randomizations are averaged. The amplitude reveals the probability to have a carry and hence the most significant bits of each word of the scalar are recovered.

The remainder of the unknown bits are recovered using the baby-step, giant-step method [Sha71].

Attack Context:

- **Key recovery**: independent bits followed by a baby-step, giant-step method to recover the missing bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the scalar randomization technique and the word size of manipulated integers,
- **Implementation Specificity**: group scalar randomization or additive splitting countermeasure,
- **Number of Executions Needed**: multiple,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

8.3.2 Additive Splitting [CJ01, §4.2]

Clavier and Joye proposed another method to randomize the scalar. Instead of computing $[k]P$, one can compute $Q = [k-r]P + [r]P$ with a random r.

This countermeasure doubles the cost of the execution since two ECSMs are performed. Another solution is to compute $[k-r]P$ and $[r]P$ in parallel. In this case, this doubles the memory required. Finally, one can use a multi-exponentiation method such as the regular Shamir's trick (Algorithm 23) with $S = P$ and the scalars $k-r$ and r.

Cost:

- 1 $\text{ECSM}_{n,n}$, 1 ECADD_n, 1 ADD_n, 1 RNG_n, 1 MEM_n in the case of a sequential computation
- 1 ECADD_n, 1 ADD_n, 1 RNG_n, and as many MEM_n needed for one $\text{ECSM}_{n,n}$ in the case of a parallel computation
- 1 ADD_n, 1 RNG_n in the case of the regular Shamir's trick (Algorithm 23)

8.3.2.1 Carry Leakage Attack [FRVD08]

In the Additive Splitting countermeasure, the scalar is subtracted to a random value. The method described in Section 8.3.1.1 can be performed during the computation of $k-r$. Indeed, as explained in Section 2.3, subtraction is generally performed using the two's complement. We compute $k-r$ as $k+\bar{r}+1$. This is exactly the scenario considered in Section 8.3.1.1 with $\bar{r}+1$ instead of $r\#E$.

8.3.2.2 Combined Attacks against Additive Splitting [MV06]

This attack of Muller and Valette consists in combining two attacks to target the Additive Splitting countermeasure.

The bits repartition of $k-r$ and r are highly correlated to the bits repartition of k. Denote by $r^{(j)}$ the random value r for the j^{th} split. A statistical analysis of the values $(k-r^{(j)})_i$ and $(r^{(j)})_i$ of different computations j can reveal k_{i+1}.

Different combined attacks are proposed to recover the statistical repartition of $(k-r^{(j)})_i$ and $(r^{(j)})_i$:

- combining two C or M Safe-Errors (see Sections 8.2.1.1 and 8.4 respectively), which is called second order Safe-Errors attack, or
- combining a C or M Safe-Error and an address-bit DSCA (see Section 8.9).

The Additive Splitting method resists to the three attacks if they are performed separately.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM, knowledge of the location of the field arithmetic module (C Safe-Error) or the memory blocks (M Safe-Error),

- **Implementation Specificity**: same addresses values on different executions, group scalar randomization or additive splitting,
- **Number of Executions Needed**: multiple,
- **Input Access**: unnecessary,
- **Output Access**: knowledge of the validity,
- **Fault Model**: any on the field arithmetic module (C Safe-Error) or data randomization on a single memory block of size n (M Safe-Error).

8.3.3 Multiplicative Splitting [TE02]

Trichina and Belleza propose another scalar randomization technique. It consists in computing $Q = [k']S$, with

- $S = [r]P$,
- $k' = kr^{-1} \bmod \#E$,

with r a random integer of small size (32 bits seems a good trade of between security and efficiency).

The countermeasure brings an additional ECSM with a 32-bit scalar[6].

Cost: 1 $\text{ECSM}_{32,n}$, 1 MUL_n, 1 mINV_n, 1 RNG_{32}, 1 MEM_{32}

8.3.4 Euclidean Splitting [CJ03, §4]

Ciet and Joye suggest to compute $Q = [k_1]P + [k_2]S$, with

- $S = [r]P$
- $k_1 = k \bmod r$
- $k_2 = \lfloor k/r \rfloor$

r being a random integer half the size of k. The result of the ECSM is the same because $[k_1]P + [k_2]S = [k \mod r]P + [\lfloor k/r \rfloor]([r]P) = [k \mod r + \lfloor k/r \rfloor r]P = [k]P$.

This scalar randomization countermeasure increases the cost of the execution of 50%: three ECSMs are performed with scalars having half the size of k. Another solution is to use the regular Shamir's trick (Algorithm 23) for the computation of $[k_1]P + [k_2]S$. In this case, there is no additional cost[7]: two ECSMs are performed (one to compute $S = [r]P$ and one to compute $[k_1]P + [k_2]S$) with scalars half the size of the initial scalar.

Cost:

- 1 $\text{ECSM}_{n/2,n}$, 1 ECADD_n, 1 DIV_n, 1 $\text{RNG}_{n/2}$, 1 $\text{MEM}_{n/2}$, 3 MEM_n
- 1 $\text{RNG}_{n/2}$, 1 DIV_n in the case of the use of the regular Shamir's trick

[7] A Jacobian to affine coordinates conversion of the point S is sometimes needed in the case where the base point needs to be in affine coordinates.

8.3.5 Point Blinding [Cor99, §5.2]

The countermeasure, by Coron, consists in computing $Q = [k](P + R)$ instead of $[k]P$, with R a pseudo-random point. The chip returns $Q - [k]R$. R and $S = [k]R$ are pre-computed and stored in the chip. After each ECSM, R and S are updated by computing $R \leftarrow (-1)^t 2R$ and $S \leftarrow (-1)^t 2S$ with t randomly chosen in $\{0, 1\}$.

The pre-computation of a random point is required because the dynamic generation of a random point is very costly. Indeed this can be done by generating a random number x, then computing the square root of $x^3 + ax + b$ (a, b being the curve parameters). Computing a square root on a limited resources system is costly.

This countermeasure was improved in [IIT04] and later in [MMM04]. The authors proposed to modify the ECSM to gradually subtract the random point R. With this improvement, the pre-computation of $[k]R$ is not necessary anymore (see Algorithm 24).

Later, Boscher, Handshuh and Trichina propose another blinded exponentiation algorithm for RSA implementations [BHT09]. We adapted it for ECC in Algorithm 25.

The countermeasure prevents the CSCA since the intermediate values are randomized by the pseudo-random point. It prevents the classical Timing attack (Section 8.1) as well.

Cost:

- 2 ECADD_n, 2 ECDBL_n, 6 MEM_n for the initial countermeasure [Cor99, §5.2]
- 1 ECADD_n, 1 ECDBL_n in the case of the use of the BRIP ECSM (Algorithm 24) or Boscher et al.'s algorithm (Algorithm 25)

8.3.6 Random Projective Coordinates [Cor99, §5.3]

A point $P = (X, Y, Z)$ in Jacobian coordinates is equivalent to any point (r^2X, r^3Y, rZ), with $r \in \mathbb{F}_p^*$. Coron suggests to randomize the base point at the beginning of the ECSM by choosing a random nonzero r.

If the base point must remain in affine coordinates for efficiency reasons (e.g. using the mixed coordinates for Algorithm 19), the randomization can be applied on the other points instead before the main loop of the ECSM (e.g. R_0, R_1 in Algorithm 19).

The countermeasure prevents the CSCA since the coordinates are randomized. It prevents the classical Timing attack (Section 8.1) as well.

Cost: 3 mMUL_n, 1 mSQR_n, 1 RNG_n

8.3.7 Random Curve Isomorphism [JT01]

Given Definition 1.4, elliptic curves $E\colon y^2 = x^3 + ax + b$ and $E'\colon y^2 = x^3 + a'x + b'$ are isomorphic if and only if there exists $u \in \mathbb{F}_p^*$ such that $u^4a' = a$ and $u^6b' = b$. The isomorphism φ is defined as:

$$\varphi\colon E \xrightarrow{\sim} E', \begin{cases} \mathcal{O} & \rightarrow & \mathcal{O} \\ (x, y) & \rightarrow & (u^{-2}x, u^{-3}y) \end{cases}$$

The countermeasure, introduced by Joye and Tymen, consists in computing the ECSM on a random curve E' instead of E. The base point is therefore randomized, as well as the parameters

a, b of the curve. Therefore, the CSCA cannot be applied. The implementation is also protected against the classical Timing attack (Section 8.1).

Cost: 8 mMUL_n, 2 mSQR_n, 1 mINV_n, 1 MEM_n, 1 RNG_n

8.4 M Safe-Error [YJ00]

M Safe-Error attacks, brought out by Yen and Joye, exploit the fact that a fault on a memory block is cleared only if the scalar meets some condition. For example, in Algorithm 21, the attacker injects an error on R_1 just after the computation of the addition and just before the storage of the result in R_{1-k_i} ([8]). The fault will be cleared only if $k_i = 0$.

The attack can target a single bit at a time. Therefore, if a scalar randomization method is used, the attack can no longer be done.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the memory blocks, knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: same addresses values on different executions,
- **Number of Executions Needed**: n,
- **Input Access**: unnecessary,
- **Output Access**: knowledge of the validity,
- **Fault Model**: data randomization on a single memory block of size n.

8.5 Invalid Point Attack [BMM00, §4]

Biehl, Meyer and Müller introduced the first Weak Curve Attack on ECC.

A fault is injected on the x coordinate of the base point $P = (x_P, y_P)$ yielding the erroneous point $\tilde{P} = (x_{\tilde{P}}, y_P)$. Instead of lying on the strong given elliptic curve $E\colon y^2 = x^3 + ax + b$, $\tilde{P}$ lies on a weak curve $\tilde{E}\colon y^2 = x^3 + ax + \tilde{b}$ for some $\tilde{b} \in \mathbb{F}_p$. Indeed, the parameter b of the initial curve is generally not involved in elliptic curve operations.

The ECSM is run with the base point $\tilde{P} \in \tilde{E}$ and the attacker recovers the result $\tilde{Q} = (x_{\tilde{Q}}, y_{\tilde{Q}}) = [k]\tilde{P}$. The attacker can deduce the value of $\tilde{b}$ with $\tilde{b} = y_{\tilde{Q}}^2 - x_{\tilde{Q}}^3 - ax_{\tilde{Q}}$.

The faulted value $x_{\tilde{P}}$ is deduced by solving the equation $y_P^2 = x^3 + ax + \tilde{b}$ on x [CJ05]. This polynomial has two or three roots. The correct candidate is the one having the most bits matching those of x_P.

The attacker can solve the ECDLP on the weak curve $\tilde{E}$ to recover k mod $ord(\tilde{P})$.

[8] This attack needs a more details on the implementation.

Attack Context:

- **Key recovery**: each time, an ECDLP on $\tilde{E}$ is performed to recover $k \bmod ord(\tilde{P})$; the full key is then recovered using the CRT,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the base point's memory blocks, the ECSM,
- **Implementation Specificity**: same addresses of the coordinates' base point on different executions,
- **Number of Executions Needed**: less than n (it depends on the order of $\tilde{P}$ on the weak curve $\tilde{E}$),
- **Input Access**: known,
- **Output Access**: known,
- **Fault Model**: data randomization on a single memory block of size n.

8.5.1 Output Point Validity [BMM00]

Biehl, Meyer and Müller suggest to verify that the computed point lies on the elliptic curve given. That is, given $Q = (x_Q, y_Q)$ and the curve equation $y^2 = x^3 + ax + b$, verify that the equality $y_Q^2 = x_Q^3 + ax_Q + b$ is satisfied. If not, no output is given.

Cost: 2 mMUL_n, 2 mSQR_n, 4 mADD_n, 1 MEM_n

8.6 Classical Differential Fault Attack [BMM00, §5]

Biehl et al. introduced the first DFA on ECC with the Right-to-Left Double-and-Add method (Algorithm 15).

First, a correct result $Q = [k]P$ is recovered. A second ECSM is run with the same scalar and the same input. Denote by Q_i the value of R_0 at the end of iteration i of Algorithm 15: $Q_i = [(k_{i-1}, \ldots, k_0)_2]P$. The attacker injects a fault on a few bits on Q_i yielding the wrong value $\tilde{Q}_i$. She recovers the erroneous result $\tilde{Q}$.

If we denote $k^{(i)} = (k_{n-1}, \ldots, k_i)_2$, we have $Q = Q_i + [2^i k^{(i)}]P$ and $\tilde{Q} = \tilde{Q}_i + [2^i k^{(i)}]P$. The attacker tries all possible values of $k^{(i)}$ to generate $Q_i = Q - [2^i k^{(i)}]P$ and $\tilde{Q}_i = \tilde{Q} - [2^i k^{(i)}]P$. The correct hypothesis of $k^{(i)}$ is the one where Q_i and $\tilde{Q}_i$ differ from only a few bits.

This attack can be iterated to recover the next bits, and it can be adapted for other ECSMs.

Remark 8.4. $\tilde{Q}_i$ does not lie on the elliptic curve. Biehl et al. argued that this is not an issue. The computation of $\tilde{Q}_i = \tilde{Q} - [2^i k^{(i)}]P$ can be performed with elements in $\mathbb{F}_p^2$ not lying on the same elliptic curve. They call it *pseudo-addition* [BMM00].

The countermeasures described in Section 8.3 against CSCA, consisting in randomizing the points or the scalar, thwart the attack. Indeed, the attacker cannot guess the intermediate points $\tilde{Q}_i$ and Q_i anymore.

In addition, this attack seems feasible only if the affine coordinates are used. Indeed, the fault induced to Q_i is done on one or several point's coordinates. If the Jacobian coordinates are used, the attacker needs to compute the same representatives of Q_i and $\tilde{Q}_i$ that actually

occurred during the execution of the ECSM. The attacker cannot perform the pseudo-additions to generate Q_i and $\tilde{Q}_i$ since she does not know the Jacobian coordinates of the result Q and $\tilde{Q}$, but only the affine coordinates. It is not clear how to take advantage of the fault in Jacobian or homogeneous coordinates.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the memory blocks, knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: affine coordinates formulæ, same addresses of the coordinates' intermediate point on different executions,
- **Number of Executions Needed**: less than n,
- **Input Access**: known and constant,
- **Output Access**: known,
- **Fault Model**: data randomization on a single register.

8.6.1 Output Point Validity [BMM00]

The method described in Section 8.5.1 was presented to prevent both the Invalid Point and Differential Fault Attacks of Sections 8.5 and 8.6 respectively.

8.7 Big Mac Attack [Wal01]

The Big Mac attack was introduced by Walter against sliding window methods on RSA implementations, with unknown inputs [Wal01]. A single trace is analysed which makes it a Horizontal Side-Channel Attack.

The attack is based on a method to detect if two multiplications share a common operand by comparing their power trace. This method was described in Section 8.2.3.2 and used for our horizontal attack against the Side-Channel Atomicity countermeasure.

If we take the example of the Left-to-Right sliding window NAF method (Algorithm 16), the comparison of the multiplications can be used to identify which point P_u or $-P_{-u}$ is being added to Q at iteration i.

The method can be applied at each iteration to deduce the whole scalar with a single ECSM. Since no leakage model is needed, this attack works on unknown and/or randomized input. This attack has been extended to the Square-and-Multiply method in [BJPW13a]. Given our definition of a same-values attack, the Big Mac attack is in fact a SVA.

Remark 8.5. The Big Mac was introduced to target RSA implementations. The success of the attack depends on the size of the manipulated integers: the longer the used integers are, the higher the success rate is [Wal01, BJPW13a]. In ECC, the integers are shorter than RSA[9]. However, more integers and operations are involved during a doubling or addition of points.

[9]For a 128 bits security, ECC must use 256-bit integers length, while RSA must use 3072-bit integers.

The attack can theoretically be applied on ECC but no practical experiment has been reported. In Section 8.2.3.2, we presented an attack based on the Big Mac principle with experimental results. This attack is however presented on a specific implementation which permits to compare many (fourteen) multiplications to balance the small size of the integers.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: full knowledge of all algorithms, knowledge of the architecture of the modular arithmetic operator,
- **Implementation Specificity**: wordwise method of the modular multiplication,
- **Number of Executions Needed**: 1,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

8.7.1 Multiplication with Random Permutation [CFG+10]

Clavier, Feix, Gagnerot, Rousselet and Verneuil introduced the Multiplication with Random Permutation countermeasure [CFG+10]. It consists in randomizing the order of the manipulation of the words during a long multiplication. For example, in Algorithm 8, it consists in randomizing the order of both loops (lines 5 and 8) with two random permutations in $[0, m[$ (m being the word number of the manipulated integers).

The construction of $s_1^{(j)}, 0 \leq j < m$ is no longer possible for the Big Mac attack. The countermeasure is also efficient against the horizontal SVA attacks.

Guessing both random permutations is not possible: there is $(m^2)!$ possibilities. However, a drawback of the countermeasure was noticed in [BJPW13a]. Bauer, Jaulmes, Prouff and Wild proved that the attack is still possible when guessing only one random permutation within the two. That reduces the possibilities to $m!$, which is possible for $m \leq 16$. This is due to the fact that the proposed multiplication method in [CFG+10] still follows a "schoolbook like" method (namely, the same word of A is used during the j loop at lines 8 to 10 in Algorithm 8).

The authors of [BJPW13a] suggest another method to correct this drawback: a single loop is performed where the words of A and B are randomly chosen with a random permutation of size m^2+2m. $2m$ additional word multiplication are performed during a long integer multiplication. Also, a second permutation of size $2m+1$ is required to avoid attacks in the carry propagation treatment. Since the modular multiplication described in Section 2.4 requires $2m^2$ word multiplications with both multiplication (m^2) and reduction (m^2), a modular multiplication with the countermeasure roughly costs $1\ \mathrm{RPG}_{m^2+2m} + 1\ \mathrm{RPG}_{2m+1} + (1 + \frac{1}{m})\ \mathrm{mMUL}_n$.

However, an adjustment of the module performing the operation described by Equation (2.1) is required since the addition with another word and the addition with the carry are performed independently. Moreover, as pointed in [CFG+10], it remains difficult to design the countermeasure in hardware due to the numerous permutations and atomic operations. The real cost might be higher than $1\ \mathrm{RPG}_{m^2+2m} + 1\ \mathrm{RPG}_{2m+1} + \frac{1}{m}\ \mathrm{mMUL}_n$.

Cost: 1 RPG_{m^2+2m}, 1 RPG_{2m+1}, $\frac{1}{m}$ mMUL_n per modular multiplication (with $m = \lceil n/w \rceil$ and w the word size of the architecture).

8.8 Horizontal Correlation Side-Channel Analysis [CFG+10]

As opposed to the Big Mac attack, the Horizontal Correlation Side-Channel Analysis of Clavier, Feix, Gagnerot, Roussellet and Verneuil was initially presented with a leakage model.

The attacker collects a single trace of an ECSM. The attack is recursive. Like the CSCA (see Section 8.3), the attacker guesses intermediate variables by making an assumption on the current bit. The attacker computes the intermediate values v_j involving during the multiplication of the supposed variables. If Algorithm 8 is used, these values can correspond to the words of the inputs, *i.e.* $a[i], b[j]$, but also all intermediate values such as $a[i] \times b[j]$, $q \times p[j]$ for $(i, j) \in [0, m[^2$. A leakage model $\mathbf{m}$ (e.g. the Hamming weight) is applied to those values, yielding a random variable Y. She performs a correlation or applies another statistical tool between the points of interest[10] of the trace of the very modular multiplication she had guessed the inputs, yielding the random variable X, and Y (see Figure 8.13). The coefficient is high if her guess is correct.

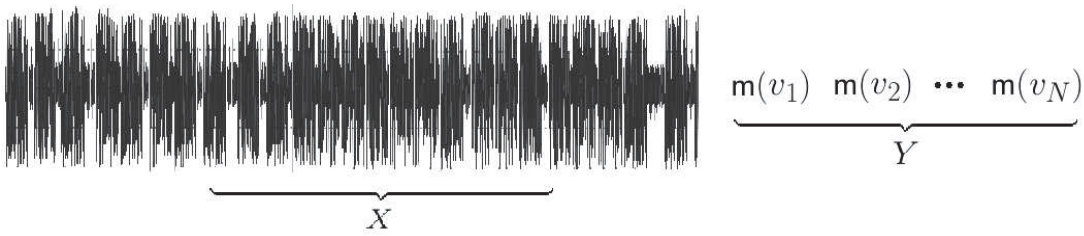


Figure 8.13: Construction of random variables for Horizontal Side-Channel Analysis with a leakage model

This attack has been extended to more exponentiation methods in [BJPW13a]. As opposed to the Big Mac attack, this attack requires the knowledge of the input. Therefore, randomizing the input, such as the Point Blinding (Section 8.3.5), the Random Projective Coordinates (Section 8.3.6) and the Random Curve Isomorphism (Section 8.3.7), thwarts the attack, only if the random is large enough to prevent a guess with a brute force approach [CFG+10].

Remark 8.6. Remark 8.5 concerning the Big Mac attack holds here: the Horizontal Correlation SCA was presented on RSA and no practical experiment has been reported on ECC due to the difference of the size of the manipulated integers.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: full knowledge of all algorithms, knowledge of the architecture of the modular arithmetic operator,
- **Implementation Specificity**: wordwise method of the modular multiplication,

[10] That is where the supposed values are possibly manipulated.

- **Number of Executions Needed**: 1,
- **Input Access**: known,
- **Output Access**: unnecessary.

8.8.1 Multiplication with Random Permutation [CFG+10]

The Random Permutation countermeasure described in Section 8.7.1 was in fact first presented against the Classical Horizontal SCA.

The classical Horizontal Attack, which consists in making assumptions on the intermediate values during the modular multiplication can no longer be applied. Indeed, the values of the random variables X, Y of Section 8.8 must be in the same order.

8.9 Address-bit DSCA [IIT02]

Address-bit Differential Side-Channel Analysis (Address-bit DSCA), introduced by Itoh, Izu and Takenaka, is a vertical attack exploiting the manipulation of addresses rather than data.

In most ECSMs, the manipulation of data depends only on a few bits of the scalar. For example, in Algorithm 21, the point R_1 is doubled only if $k_i = 1$.

The attack consists in detecting if the manipulated addresses during the doubling at iteration i are the same as the ones of a reference. The reference can be the doubling of the first iteration: the attacker supposes that $k_{n-2} = 1$ ([11]). The random variable X is constructed where the addresses of R_1 are manipulated during the doubling of the first iteration.

Then, the attacker constructs the random variable Y where the addresses of R_{k_i} are manipulated during the doubling at iteration i. In [IIT02], the authors suggest to perform a difference of means between X and Y. An another statistical method (e.g. a correlation or the Euclidean distance) can be used instead. Figure 8.8 illustrates the construction of the random variables.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM,
- **Implementation Specificity**: same addresses values on different executions,
- **Number of Executions Needed**: multiple,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary.

[11] If her guess is incorrect, at the end of the attack, she will recover $\bar{k}$ instead of k. The correct value of the scalar is trivially recovered.

8.9.1 Random Register Address [IIT03]

The authors that introduced the Address-bit DSCA proposed a countermeasures to thwart it. It consists in randomizing the addresses at each iteration of the ECSM. A flaw of the countermeasure has been showed by Izumi, Ikegami, Sakiyama and Ohta and the countermeasure was improved [IISO10]. In [IIT03], an extra point is necessary for the Left-to-Right Double-and-Add always method. We propose an alternative solution illustrated in Algorithm 32 without any extra point. For the Montgomery Ladder, an extra point in necessary [IISO10].

Algorithm 32 Left-to-Right Double-and-Add always with Random Address

Input: $k = (1, k_{n-2}, \ldots, k_0)_2$, P
Output: $[k]P$
$r \xleftarrow{\mathcal{R}} [0, 2^n[$
$R_0 \leftarrow P, R_1 \leftarrow P$
for $i = n-2$ **downto** 0 **do**
  $R_{r_i} \leftarrow 2R_{r_{i+1}}$
  $R_{1-(r_i \oplus k_i)} \leftarrow R_{r_i} + P$
return R_{r_0}

The countermeasure prevents the address-bit DSCA, described in Section 8.9. It also prevents M Safe-Errors (Section 8.4) since the addresses are randomized.

Cost:

- 1 RNG_n, 1 MEM_n for the Left-to-Right Double-and-Add always method,
- 1 RNG_n, 4 MEM_n for the Montgomery Ladder.

8.10 Doubling Attack [FV03]

The doubling attack of Fouque and Valette relies on the power consumption comparison of two ECSMs with the base point P and $[2]P$, respectively. This is a vertical attack (with only two traces).

The same values occur in the two ECSMs only if the scalar meets some condition (e.g. the current bit is 0) and therefore a collision of traces can be detected. In [FV03], the authors suggest to perform a difference between the traces to detect a collision. The doubling attack is illustrated in Figure 8.14.

Due to the birthday paradox, the doubling attack can be performed even if the Group Scalar Randomization (Section 8.3.1) or the Multiplicative Splitting (Section 8.3.3) is used [FV03].

The Point Blinding countermeasure (Section 8.3.5) is vulnerable as well [FV03]. This is because the pseudo-random point R, intended to randomize the base point P, is updated as $R \leftarrow (-1)^t 2R$, with $t \in [0, 1]$, after each ECSM. There is a probability of $1/2$, that the attack still works.

The doubling attack can be mounted only if elliptic curve formulæ are implemented using the affine coordinates. Indeed, in Jacobian coordinates, the base point $[2]P$ of the second ECSM is first given in affine coordinates. Same projective points will occur during the computation

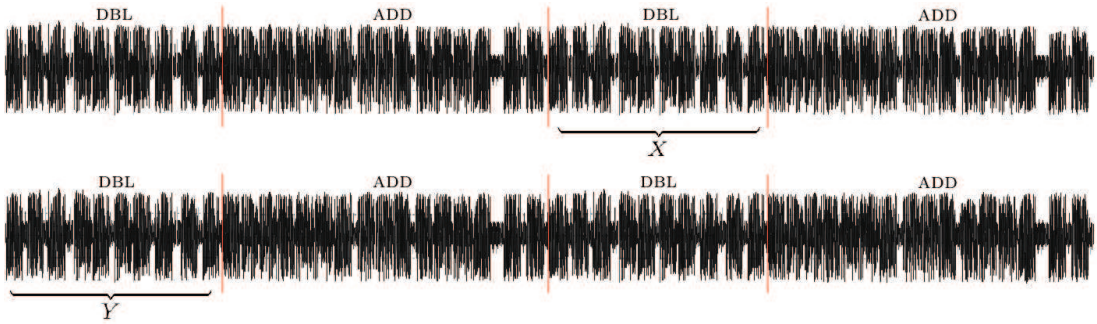


Figure 8.14: Construction of random variables for the doubling attack

$[k]P$ and $[k]([2]P)$ but with different representatives. We recall that, in Jacobian coordinates, there is the equivalence relation $(X, Y, Z) \sim (r^2X, r^3Y, rZ)$ with $r \in \mathbb{F}_p^*$.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: affine coordinates formulæ,
- **Number of Executions Needed**: 2,
- **Input Access**: 1 known and 1 chosen,
- **Output Access**: unnecessary.

8.11 Refined Side-Channel Analysis [Gou03]

The Refined Side-Channel Analysis (RSCA) of Goubin is based on the occurrence of the particular point $P_0 = (0, y)$ during the ECSM. It is a SSCA.

The attacker chooses the base point P such that the special point P_0 will occur under some assumption (for example the current targeted bit is 0). The computation of such a point P is performed as follows, with the example of the Double-and-Add always method (Algorithm 19).

Suppose that the attacker already knows the $n - i - 1$ leftmost bits of the fixed scalar $k = (k_{n-1}, \ldots, k_0)_2$ and tries to recover k_i. If the base point P is chosen such that $P = [(k_{n-1}, \ldots, k_{i+1}, 1)_2^{-1} \bmod \#E]P_0$, P_0 will be doubled at iteration $i - 1$ only if $k_i = 1$.

The doubling of the point P_0 can be detected by observing the trace, as shown in Figure 8.15.

Obviously, the particular point is not randomized neither by the Random Projective nor by the Random Curve Isomorphism countermeasures (see Sections 8.3.6 and 8.3.7).

Scalar randomization techniques help prevent the RSCA since the recursive process is broken. However, an attacker can target several bits at a time. Several bits of the randomized scalar can be recovered and reveal some information on the initial scalar.

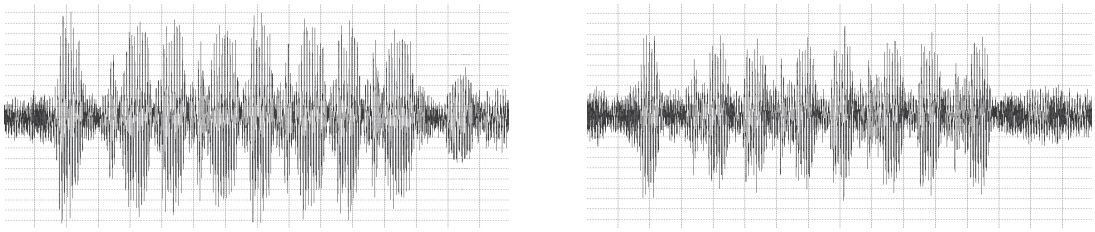


Figure 8.15: Power consumption of 256-bits modular multiplications of two random operands (left curve) and a random operand and zero (right curve)

The Point Blinding described in Section 8.3.5 thwarts the attack because the point $P_0 + R$ with an unknown pseudo-random point R will occur instead of P_0.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: must contain a point of the form $(0, y)$ for some $y \in \mathbb{F}_p$,
- **Implementation Access**: knowledge of the ECSM,
- **Implementation Specificity**: none,
- **Number of Executions Needed**: n,
- **Input Access**: chosen,
- **Output Access**: unnecessary.

8.11.1 Isomorphism Shifting [DGH+12]

We present our countermeasure thwarting the RSCA. The basic principle is to use an isomorphism to transform the base point into the inconvenient point of the RSCA, namely $(0, y)$. We "control" this point and its apparition does not reveal anything about the scalar. Moreover, on some ECSMs, the extra cost of the countermeasure is negligible and even negative thanks to the zero value.

Transformation of the base point using an isomorphism. With Definition 1.4, the elliptic curves E and E' given by the equations

$$\begin{array}{ll} E: & y^2 = x^3 + a_4 x + a_6 \\ E': & y^2 = x^3 + a'_2 x^2 + a'_4 x + a'_6 \end{array}$$

are isomorphic over $\mathbb{F}_p$ if and only if there exist $u \in \mathbb{F}_p^*$ and $r \in \mathbb{F}_p$ such that the change of variables

$$(x, y) \rightarrow (u^{-2}(x - r), u^{-3} y)$$

transforms equation E into equation E' with:

$$\left\{ \begin{array}{lcl} u^2 a'_2 & = & 3r \\ u^4 a'_4 & = & a_4 + 3r^2 \\ u^6 a'_6 & = & a_6 + r a_4 + r^3 \ . \end{array} \right.$$

This isomorphism is a particular case of Definition 1.4 with $s = t = 0$. Given the base point $P = (x_P, y_P)$ we can choose $u = 1$ and $r = x_P$. The isomorphism φ is

$$\varphi\colon E \xrightarrow{\sim} E', \begin{cases} \mathcal{O} & \to & \mathcal{O} \\ (x, y) & \to & (x - x_P, y) \end{cases}$$

and transforms P into $\varphi(P) = P' = (0, y_P)$. Applying the isomorphism on the curve costs $2\ \text{mMUL}_n + 1\ \text{mSQR}_n + 5\ \text{mADD}_n$. Applying the isomorphism on a point costs only $1\ \text{mADD}_n$. It costs also $1\ \text{mADD}_n$ for the isomorphism inverse. Two extra memory blocs of n bits are also required to store a'_2 and x_P. The transformation over $\mathbb{R}$ is illustrated in Figure 8.16.

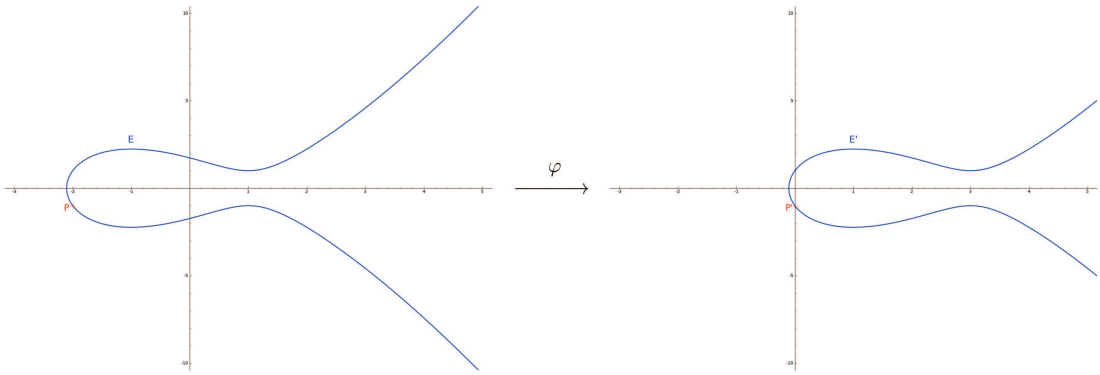


Figure 8.16: Isomorphism Shifting with $E\colon y^2 = x^3 - 3x + 3$ and $E'\colon y^2 = x^3 - 6x^2 + 9x + 1$.

Modifications of the elliptic curve formulæ. The isomorphic curve E' is not in its Weierstraß equation. The formulæ given in Section 3.1 do not apply with the curve E'. Extra fields operations are required using the general formulæ in affine coordinates of Section 1.2 (with $a_1 = a_3 = 0$).

However, the x coordinate of P' is equal to zero. We can remove the unnecessary field operations when adding P' or $-P'$. If P' is the base point, P' and $-P'$ are involved at each iteration in some ECSMs, such as the Left-to-Right Double-and-Add methods (Algorithms 14 and 19) and the Montgomery Ladder with co-Z formulæ (Algorithm 22).

We give the adapted formulæ that can be used for the three ECSMs listed above. The input and output points lie on the curve $E'\colon y^2 = x^3 + a'_2x^2 + a'_4x + a'_6$. The value $a'_2Z_3^2$ is always computed (Z_3 being the Z coordinate of the output point), we can add it in the coordinates. We denote it by U.

Algorithm 33 ECADD-ISO-SHIFTING

Input: $P' = (0, Y_1, Z_1, U_1 = a'_2 Z_1^2), Q' = (X_2, Y_2, Z_2, U_2 = a'_2 Z_2^2)$, elliptic curve parameter a'_2
Output: $P' + Q'$
$B \leftarrow X_2 Z_1^2;\ C \leftarrow Y_1 Z_2^3;\ D \leftarrow Y_2 Z_1^3$
$F \leftarrow D - C$
$Z'_3 \leftarrow Z_1 Z_2 B$
$U_3 \leftarrow a'_2 Z_3^2$
$X_3 \leftarrow F^2 - B^3 - U_3$
$Y_3 \leftarrow -F X_3 - C B^3$
return (X_3, Y_3, Z_3, U_3)

Like the classical formulæ, if P' is in affine coordinates, we can save four multiplications and one square (mixed addition: mECADD-ISO-SHIFTING). If Z_1^2 and Z_1^3 are pre-computed, one multiplication and one square are saved (re-addition: reECADD-ISO-SHIFTING).
Cost (ECADD-ISO-SHIFTING): 11 mMUL_n, 5 mSQR_n, 5 mADD_n, 5 MEM_n
Cost (mECADD-ISO-SHIFTING): 7 mMUL_n, 4 mSQR_n, 5 mADD_n, 3 MEM_n
Cost (reECADD-ISO-SHIFTING): 10 mMUL_n, 4 mSQR_n, 5 mADD_n, 5 MEM_n
For the three additions, the difference is 1 $\text{mSQR}_n - 1$ $\text{mMUL}_n - 2$ mADD_n compared with ECADD, mECADD and reECADD.

Algorithm 34 ECDBL-ISO-SHIFTING

Input: $Q' = (X_1, Y_1, Z_1, U_1 = a'_2 Z_1^2)$, elliptic curve parameters a'_2, a'_4
Output: $2Q'$
$A \leftarrow 2Y_1^2$
$B \leftarrow 2AX_1;\ C \leftarrow 3X_1^2 + 2X_1 U_1 + a'_4 Z_1^4;\ D \leftarrow 2A^2$
$Z_3 \leftarrow 2Y_1 Z_1$
$U_3 \leftarrow a'_2 Z_3^2$
$X_3 \leftarrow C^2 - 2B - U_3$
$Y_3 \leftarrow C(B - X_3) - D$
return (X_3, Y_3, Z_3, U_3)

If (X_3, Y_3, Z_3, U_3) is used later for re-addition, the computation of Z_3^2 and Z_3^3 needs one extra multiplication[12] (reECDBL-ISO-SHIFTING). If $a'_4 Z_1^4$ is pre-computed (modified formulæ: modECDBL-ISO-SHIFTING), two squares are saved, and it needs one extra addition for the computation of $a'_4 Z_3^4$. The use of both modified coordinates and re-addition is also given (mod-reECDBL-ISO-SHIFTING).
Cost (ECDBL-ISO-SHIFTING): 7 mMUL_n, 6 mSQR_n, 14 mADD_n, 3 MEM_n
Cost (reECDBL-ISO-SHIFTING): 8 mMUL_n, 6 mSQR_n, 14 mADD_n, 4 MEM_n
Cost (modECDBL-ISO-SHIFTING): 7 mMUL_n, 4 mSQR_n, 15 mADD_n, 4 MEM_n
Cost (mod-reECDBL-ISO-SHIFTING): 8 mMUL_n, 5 mSQR_n, 15 mADD_n, 4 MEM_n

The extra cost is 3 $\text{mMUL}_n + 3$ mADD_n, 3 $\text{mMUL}_n - 1$ $\text{mSQR}_n + 3$ mADD_n, 3 $\text{mMUL}_n + 3$ mADD_n compared with the general doubling, the doubling for re-addition, the doubling in modified coordinates and the doubling in both modified coordinates and re-addition, respectively.

For the Montgomery Ladder using co-Z formulæ (Algorithm 22), the cost is even smaller because no doubling is performed. We give the adapted formulæ below.

[12] This is because Z_3^2 is computed anyway for U_3.

Algorithm 35 co-Z addition and update with the isomorphism shifting (ZADDU-ISO-SHIFTING)

Input: $P' = (X_1, Y_1, Z, U = a'_2 Z^2), Q' = (0, Y_2, Z, U)$
Output: (R', S') with $R' = P' + Q'$ and $S' = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z, a'_2(\lambda Z)^2)$ with $\lambda = X_1$
$C \leftarrow X_1^2$
$W_1 \leftarrow X_1 C;\ Z_3 \leftarrow Z X_1;\ U_3 \leftarrow UC$
$D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1 W_1$
$X_3 \leftarrow D - W_1 - U_3$
$Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
$X_4 \leftarrow W_1$
$Y_4 \leftarrow A_1$
return $((X_3, Y_3, Z_3, U_3), (X_4, Y_4, Z_3, U_3))$

Like the classical co-Z addition and update, the computation of the Z coordinate is not necessary and one multiplication is saved (ZADDU'-ISO-SHIFTING).
Cost (ZADDU-ISO-SHIFTING): 5 mMUL_n, 2 mSQR_n, 5 mADD_n, 1 MEM_n
Cost (ZADDU'-ISO-SHIFTING): 4 mMUL_n, 2 mSQR_n, 5 mADD_n, 1 MEM_n

For both formulæ, the gain is 2 mADD_n compared with ZADDU and ZADDU'.

Algorithm 36 conjugate co-Z addition with the isomorphism shifting (ZADDC-ISO-SHIFTING)

Input: $P' = (X_1, Y_1, Z, U = a'_2 Z^2), Q' = (X_2, Y_2, Z, U)$ such that $x_{P'-Q'} = 0$.
Output: (R', S) with $R' = P' + Q'$, $S' = P' - Q'$
$C \leftarrow (X_1 - X_2)^2$
$W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C;\ Z_3 \leftarrow Z(X_1 - X_2);\ U_3 \leftarrow UC$
$D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_3 \leftarrow D - W_1 - W_2 - U_3$
$Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
$Y_4 \leftarrow (Y_1 + Y_2)W_1 - A_1$
return $((X_3, Y_3, Z_3, U_3), (0, Y_4, Z_3, U_3))$

Like the classical conjugate co-Z addition, the computation of the Z coordinate is not necessary and one multiplication is saved (ZADDC'-ISO-SHIFTING).
Cost (ZADDC-ISO-SHIFTING): 7 mMUL_n, 2 mSQR_n, 10 mADD_n, 2 MEM_n
Cost (ZADDC'-ISO-SHIFTING): 6 mMUL_n, 2 mSQR_n, 10 mADD_n, 2 MEM_n

For both formulæ, the difference is 1 $\mathrm{mMUL}_n - 1$ mSQR $- 1$ mADD_n compared with ZADDU and ZADDU'. The formulæ with register allocation, is given in appendix.

An important remark is that, if 1 $\mathrm{mMUL}_n = 1$ mSQR_n, we gain three modular additions per bit for the Montgomery Ladder using co-Z formulæ. Therefore, it is a non-negligible improvement to apply the countermeasure in this condition.

Security Analysis. We explain here how the countermeasure prevents the RSCA. The elliptic curve E': $y^2 = x^3 + a'_2 x^2 + a'_4 x + a'_6$ contains exactly two points of the form $(0, y')$. Those points are $P' = (0, \sqrt{a'_6})$ and $-P' = (0, -\sqrt{a'_6})$. We give the following theorems:

Theorem 8.7. *Suppose that the Left-to-Right Double-and-Add (Algorithm 14) is performed with a scalar* $k = (k_{n-1}, \ldots, k_0)_2$*, the base point* $P' = (0, y_P)$ *and the formulæ* ECADD-ISO-SHIFTING *and* ECDBL-ISO-SHIFTING*. Let* $k^{(i)} = (k_{n-1}, k_{n-2}, \ldots, k_i)_2$*. Suppose that* $gcd(k^{(i)} \pm 1, ord(P')) = 1$ *and* $gcd(2k^{(i+1)} \pm 1, ord(P')) = 1$ *for all* $i \in [0, n-2[$*.*
Then R_0 *cannot be equal to* $\pm P'$ *at any iteration* $i \in [0, n-2[$*.*

Proof. At the beginning of iteration i with $n-2 < i \leq 0$, the point R_0 verifies $R_0 = [k^{(i+1)}]P'$.

- if $R_0 = [k^{(i+1)}]P' = P'$, then $[k^{(i+1)} - 1]P' = \mathcal{O}$ so the order of P' is $(k^{(i+1)} - 1)$, which contradicts our hypothesis on k.
- if $R_0 = [2k^{(i+1)}]P' = P'$ after the doubling, then $[2k^{(i+1)} - 1]P' = \mathcal{O}$ so the order of P' is $(2k^{(i+1)} - 1)$, which contradicts our hypothesis on k.
- if $R_0 = [k^{(i)}]P' = P'$ after the doubling, then $[k^{(i)} - 1]P' = \mathcal{O}$ so the order of P' is $(k^{(i)} - 1)$, which contradicts our hypothesis on k.
- if $R_0 = [k^{(i+1)}]P' = -P'$, then $[k^{(i+1)} + 1]P' = \mathcal{O}$ so the order of P' is $(k^{(i+1)} + 1)$, which contradicts our hypothesis on k.
- if $R_0 = [2k^{(i+1)}]P' = -P'$ after the doubling, then $[2k^{(i+1)} + 1]P' = \mathcal{O}$ so the order of P' is $(2k^{(i+1)} + 1)$, which contradicts our hypothesis on k.
- if $R_0 = [k^{(i)}]P' = -P'$ after the doubling, then $[k^{(i)} + 1]P' = \mathcal{O}$ so the order of P' is $(k^{(i)} + 1)$, which contradicts our hypothesis on k.

□

Remark 8.8. The condition $gcd(k^{(i)} \pm 1, ord(P')) = 1$ and $gcd(2k^{(i+1)} \pm 1, ord(P')) = 1$ for all $n-2 < i \leq 0$ is not binding. If l is the security parameter, then $ord(P')$ is a prime and $ord(P') \approx 2^{2l}$. This is explained in Chapter 6.

Corollary 8.1. *Suppose that the Left-to-Right Double-and-Add always (Algorithm 19) is performed with a scalar* $k = (k_{n-1}, \ldots, k_0)_2$*, the base point* $P' = (0, y_P)$ *and the formulæ* ECADD-ISO-SHIFTING *and* ECDBL-ISO-SHIFTING*. Let* $k^{(i)} = (k_{n-1}, k_{n-2}, \ldots, k_i)_2$*. Suppose that* $gcd(k^{(i)} + 1 \pm 1, ord(P')) = 1$*,* $gcd(k^{(i)} \pm 1, ord(P')) = 1$ *and* $gcd(2k^{(i+1)} \pm 1, ord(P')) = 1$ *for all* $i \in [0, n-2[$*.*
Then neither R_0 *nor* R_1 *can be equal to* $\pm P'$ *at any iteration* $i \in [0, n-2[$*.*

Proof. In addition to the previous theorem, we only need to check that the value of R_1 cannot take the value $\pm P'$. This is verified as long as $gcd(k^{(i)} + 1 \pm 1, ord(P')) = 1$. □

With the countermeasure, both ECSMs are secure against the RSCA since the inconvenient points are managed and the attacker cannot take advantage of their occurrence.

Theorem 8.9. *Suppose that the Montgomery Ladder using co-Z formulæ (Algorithm 22) is performed with a scalar* $k = (k_{n-1}, \ldots, k_0)_2$*, the base point* $P' = (0, y_P)$ *and the formulæ* ZADDU-ISO-SHIFTING *and* ZADDC-ISO-SHIFTING*. Let* $k^{(i)} = (k_{n-1}, k_{n-2}, \ldots, k_i)_2$*. Suppose that* $gcd(k^{(i)}, ord(P')) = 1$*,* $gcd(k^{(i)} \pm 1, ord(P')) = 1$ *and* $gcd(k^{(i)} + 2, ord(P')) = 1$ *for all* $i \in [0, n-1[$*.*
Then neither R_0 *nor* R_1 *can be equal to* $\pm P'$ *at the end of any iteration.*

Proof. At the end of iteration i with $n-2 < i \leq 0$, the points R_0, R_1 verify $R_0 = [k^{(i)}]P', R_1 = [k^{(i)} + 1]P'$.

- if $R_0 = [k^{(i)}]P' = P'$, then $[k^{(i)} - 1]P' = \mathcal{O}$ so the order of P' is $(k^{(i)} - 1)$, which contradicts our hypothesis on k.
- if $R_1 = [k^{(i)} + 1]P' = P'$, then $[k^{(i)}]P' = \mathcal{O}$ so the order of P' is $(k^{(i)})$, which contradicts our hypothesis on k.
- if $R_0 = [k^{(i)}]P' = -P'$, then $[k^{(i)} + 1]P' = \mathcal{O}$ so the order of P' is $(k^{(i)} + 1)$, which contradicts our hypothesis on k.
- if $R_1 = [k^{(i)} + 1]P' = -P'$, then $[k^{(i)} + 2]P' = \mathcal{O}$ so the order of P' is $(k^{(i)} + 2)$, which contradicts our hypothesis on k.

□

With the countermeasure, the ECSM is secure against the RSCA since the inconvenient points are managed.

We presented an elegant countermeasure against the RSCA using an isomorphism between curves. Because of the isomorphism, the formulæ have been reviewed.

Cost:

- $(2.5n+2)$ mMUL_n, $(0.5n+1)$ mSQR, $(2n+6)$ mADD, 4 MEM_n for the Left-to-Right Double-and-Add method (Algorithm 14)
- $(2n+2)$ mMUL_n, $(n+1)$ mSQR, $(n+6)$ mADD, 5 MEM_n for the Left-to-Right Double-and-Add always method (Algorithm 19)
- $(n+2)$ mMUL_n, $(-n+1)$ mSQR, $(-3n+6)$ mADD for the Montgomery Ladder using co-Z formulæ (Algorithm 22)

8.12 Zero Side-Channel Analysis [AT03]

The Zero Side-Channel Analysis (ZSCA) of Akishita and Takagi is an extension of the RSCA.

This attack does not only focus on a zero value in a point's coordinates but on intermediate values when performing a doubling or an addition. Such points are defined as *zero-value points*. For example, consider ECDBL with register allocation (Algorithm 39). Let $P = (x_P, y_P)$ a point such that $3x_P + a = 0$ in affine coordinates. The doubling of P in Jacobian coordinates will lead to the condition $C = 3X_P^2 + aZ_P^4 = 0$, with $X_P = x_P Z_P^2$ for some $Z_P \in F_p^*$. The ZSCA brings more possible particular points to consider.

As the RSCA, the Random Projective or the Random Curve Isomorphism countermeasures (see Sections 8.3.6 and 8.3.7) does not prevent this attack.

As the RSCA, scalar randomization techniques help prevent the ZSCA since the recursive process is broken and the Point Blinding (see Section 8.3.5) thwarts the attack.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: must contain zero-value points,
- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,

- **Implementation Specificity**: none,
- **Number of Executions Needed**: n,
- **Input**: chosen,
- **Output**: unnecessary.

8.13 Classical Same Values Side-Channel Analysis [MGD+12]

We present our vertical SCA called the classical Same Values Side-Channel Analysis (SVA). Like the RSCA and ZSCA (Sections 8.11 and 8.12), it exploits the occurrence of particular points. These points verify that, within an elliptic curve operation (e.g. an addition or a doubling), two distinct intermediate variables have the same values.

Some curves do not contain any zero-value point for performing the RSCA or ZSCA. The SVA considerably increases the number of particular points the attacker can use to mount an attack, and therefore, works on a larger set of curves.

Same-values points. We introduce the definitions of the particular points that the attacker will take advantage to perform the SVA.

Definition 8.10. Let E be an elliptic curve over $\mathbb{F}_p$ and ECDBL a doubling algorithm. A point $P = (x_1, y_1) \in E$ is a *same-values point* relative to ECDBL if, for any representative of P (i.e. $(\lambda^2 x_1, \lambda^3 y_1, \lambda)$ for all $\lambda \in \mathbb{F}_p^*$ in Jacobian coordinates), same values show up among intermediate variables during the computation of the point $2P$ using algorithm ECDBL.

Definition 8.11. Let E be an elliptic curve over $\mathbb{F}_p$ and ECADD an addition algorithm (respectively C-ECADD a conjugate addition algorithm). Points $P, Q \in E$ are said to be *same-values points* relative to ECADD (resp. to C-ECADD) if, for any representatives of P and Q, same values show up among intermediate variables during the computation of the point $P+Q$ using algorithm ECADD (resp. the computation of $P+Q$ and $P-Q$ using algorithm C-ECADD).

The same-values points depend on the elliptic curve formulæ used. The formulæ given in Section 3.1 are not accurate enough to identify the same-values points. The knowledge of the ECDBL and/or ECADD algorithms, together with register allocation is required. We will give examples of same-values points for a doubling algorithm and for an addition algorithm.

Theorem 8.12. *Let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over $\mathbb{F}_p$. Let $P = (x_1, y_1)$ a point lying on E and $R = (x_3, y_3) = [2]P$. P is a same-values point relative to* ECDBL *with register allocation (Algorithm 39 in appendix) if one of the following conditions is satisfied:*

1. $x_1 = 1$
2. $2x_1y_1^2 = (3x_1^2 + a)^2$
3. $6x_1y_1^2 = (3x_1^2 + a)^2$
4. $8x_1y_1^2 = (3x_1^2 + a)^2$
5. $10x_1y_1^2 = (3x_1^2 + a)^2$
6. $12x_1y_1^2 = (3x_1^2 + a)^2$
7. $-4y_1^4 = y_3$
8. $12y_1^4 = y_3$
9. $16y_1^4 = y_3$
10. $x_1 = 0$
11. $y_1 = 0$
12. $x_3 = 0$
13. $y_3 = 0$
14. $3x_1^2 = -a$

15. $4x_1y_1^2 = (3x_1^2 + a)^2$
16. $x_1^2 = y_1$
17. $x_1^2 = 2y_1$
18. $x_1 = -1$
19. $x_1^2 = a$
20. $-2x_1^2 + a = 0$
21. $2x_1^2 = y_1$
22. $3x_1^2 = y_1$
23. $y_1 = 1$
24. $y_1 = a$
25. $y_1 = 3x_1^2 + a$
26. $3x_1^2 = 2y_1$
27. $2y_1 = 1$
28. $2y_1 = 3x_1^2 + a$
29. $2x_1^2 = 1$
30. $2x_1^2 = a$
31. $-x_1^2 + a = 0$
32. $3x_1^2 = 1$
33. $3x_1^2 = a$
34. $3x_1^2 + a = 1$

Proof. Given a point $P = (X_1, Y_1, Z_1) = (\lambda^2 x_1, \lambda^3 y_1, \lambda)$, the relations of Theorem 8.12 must hold for any $\lambda \in \mathbb{F}_p^*$. So we must check equalities between terms with a factor λ of the same degree. Let S_i be the set of values that involve a factor λ of degree i. An analysis of Algorithm 39 gives:

- $S_1 = \{Z_1\}$,
- $S_2 = \{X_1, Z_1^2\}$,
- $S_3 = \{Y_1\}$,
- $S_4 = \{X_1^2, Y_1Z_1, 2Y_1Z_1, 2X_1^2, 3X_1^2, Z_1^4, aZ_1^4, C = 3X_1^2 + aZ_1^4\}$,
- $S_6 = \{Y_1^2, 2Y_1^2\}$,
- $S_8 = \{2X_1Y_1^2, B = 4X_1Y_1^2, C^2, X_3 + B = C^2 - B, X_3 = C^2 - 2B, B - X_3 = 3B - C^2\}$,
- $S_{12} = \{A^2 = 4Y_1^4, 2A^2 = 8Y_1^4, Y_3 + 2A^2 = C(B - X_3), Y_3 = C(B - X_3) - 2A^2\}$

Equal values can only be found in the same set. Comparing the terms from each other by set, and developing give the relations of the theorem.

□

Remark 8.13. Points satisfying one of the condition

- $x_1 = 0$ or
- $y_1 = 0$ or
- $x_3 = 0$ or
- $y_1 = 0$ or
- $y_3 = 0$ or
- $3x_1^2 = -a$ or

- $4x_1y_1^2 = (3x_1^2 + a)^2$

are zero-value points.

Theorem 8.14. *Let $E\colon y^2 = x^3 + ax + b$ be an elliptic curve over $\mathbb{F}_p$. Let $P = (x_1, y_1), Q = (x_2, y_2)$ two points lying on E and $R = (x_3, y_3) = P + Q$. P, Q are same-values points relative to* ECADD *with register allocation (Algorithm 37 in appendix) if one of the following conditions given below is satisfied:*

1. $x_1 = 1$
2. $y_1 = 1$
3. $x_2 = 1$
4. $y_2 = 1$
5. $x_1 = x_2$
6. $2x_1 = x_2$
7. $y_1(x_1 - x_2)^3 = y_3$
8. $x_1 = 0$
9. $y_1 = 0$
10. $x_1 = x_2$
11. $x_3 = 0$
12. $y_3 = 0$
13. $x_1(x_2 - x_1)^2 = -x_3$
14. $x_1(x_2 - x_1)^2 = x_3$
15. $(x_2 - x_1)^3 = (y_2 - y_1)^2$
16. $x_2 - x_1 = y_1$
17. $x_2 - x_1 = y_2$
18. $x_2 - x_1 = y_2 - y_1$
19. $y_1 = y_2$
20. $2y_1 = y_2$
21. $x_1(x_2 - x_1)^2 = (y_2 - y_1)^2$
22. $2(x_2 - x_1)^3 = (y_2 - y_1)^2$
23. $2(x_2 - x_1)^3 = (y_2 - y_1)^2 - x_1(x_2 - x_1)^2$
24. $2(x_2 - x_1)^3 = (y_2 - y_1)^2 - 2x_1(x_2 - x_1)^2$
25. $3x_1(x_2 - x_1)^2 = (y_2 - y_1)^2$
26. $x_1(x_2 - x_1)^2 = (x_1 - x_2)^3$
27. $2x_1(x_2 - x_1)^2 = (x_1 - x_2)^3$
28. $2(y_2 - y_1)^2 = 3x_1(x_2 - x_1)^2 + (x_2 - x_1)^3$
29. $2(y_2 - y_1)^2 = 3x_1(x_2 - x_1)^2 + 2(x_2 - x_1)^3$
30. $2x_3 = x_1(x_2 - x_1)^2$

Proof. Given points $P = (X_1, Y_1, Z_1) = (\lambda_1^2 x_1, \lambda_1^3 y_1, \lambda_1)$ and $Q = (X_2, Y_2, Z_2) = (\lambda_2^2 x_2, \lambda_2^3 y_2, \lambda_2)$, the relations of Theorem 8.12 must hold for any $\lambda_1, \lambda_2 \in \mathbb{F}_p^*$. So we must check equalities between terms with factors λ_1, λ_2 of the same degree. Let $S_{i,j}$ be the set of values that involve a factor λ_1 of degree i and a factor λ_2 of degree j. An analysis of Algorithm 37 gives:

- $S_{1,0} = \{Z_1\}$
- $S_{0,1} = \{Z_2\}$
- $S_{1,1} = \{Z_1 Z_2\}$
- $S_{2,0} = \{X_1, Z_1^2\}$
- $S_{3,0} = \{Y_1, Z_1^3\}$
- $S_{0,2} = \{X_2, Z_2^2\}$
- $S_{0,3} = \{Y_2, Z_2^3\}$

- $S_{2,2} = \{A = X_1Z_2^2, B = X_2Z_1^2, E = X_2Z_1^2 - X_1Z_2^2\}$
- $S_{3,3} = \{Z_3 = Z_1Z_2E, C = Y_1Z_2^3, D = Y_2Z_1^3, F = Y_2Z_1^3 - Y_1Z_2^3\}$
- $S_{4,4} = \{E^2\}$
- $S_{6,6} = \{AE^2, E^3, F^2, F^2 - E^3, X_3 + AE^2 = F^2 - E^3 - AE^2, X_3 = F^2 - E^3 - 2AE^2, AE^2 - X_3 = 3AE^2 + E^3 - F^2\}$
- $S_{9,9} = \{CE^3, Y_3 + CE^3 = F(AE^2 - X_3), Y_3 = F(AE^2 - X_3) - CE^3\}$

Equal values can only be found in the same set. Comparing the terms from each other by set, and developing give the relations of the theorem.

□

Remark 8.15. Points satisfying one of the following condition

- $x_1 = 0$ or
- $y_1 = 0$ or
- $x_1 = x_2$ or
- $y_3 = 0$ or
- $x_1(x_2 - x_1)^2 = -x_3$ or
- $x_1(x_2 - x_1)^2 = x_3$ or
- $(x_2 - x_1)^3 = (y_2 - y_1)^2$

are zero-value points.

Remark 8.16. If E and E' are both given in their reduced Weierstraß form, the isomorphism φ is defined as

$$\varphi : E \xrightarrow{\sim} E', \begin{cases} \mathcal{O} & \to & \mathcal{O} \\ (x, y) & \to & (u^{-2}x, u^{-3}y) \end{cases}$$

for some $u \in \mathbb{F}_p^*$. If P is a same-values point relative to a doubling algorithm on E, this does not imply that it is a same-values point on E'. For example, comparing the values of the set $S_2 = \{X_1, Z_1^2\}$ of Theorem 8.12, leading to the affine condition $x_1 = 1$, is not relevant anymore on E'. Indeed, if $X_1 = Z_1^2$ (P is a same-values point on E) then $u^{-2}X_1 \neq Z_1^2$ if $u \neq \pm 1$ (P' is not a same-values point on E').

Therefore, the Random Curve Isomorphism, described in Section 8.3.7, decreases the number of same-values points relative to some elliptic curve operations but does not entirely prevents the attack since some equalities still hold. For example, $2X_1^2$ can be compared to $C = 3X_1^2 + aZ_1^4$ (set S_4 of Theorem 8.12) whatever the value of u.

Taking advantage of the same-values points is similar to the RSCA and ZSCA (Sections 8.11 and 8.12) and it is recalled below.

Choosing the suitable base point. First, the attacker finds a same-values point P_{SVA} relative to the doubling formula. She then chooses a base point such that P_{SVA} will occur on a certain condition of the scalar (e.g. the current targeted bit is 1). The computation of such a point P is performed as follows, with the example of the Double-and-Add always method (Algorithm 19). It can be adapted for the other ECSMs. Assume that the attacker already knows the $n-i-1$ leftmost bits of the fixed scalar $k = (k_{n-1}, \ldots, k_0)_2$ and tries to recover k_i. The attacker computes the point $P = [(k_{n-1}, \ldots, k_{i+1}, 1)_2^{-1} \bmod \#E]P_{\text{SVA}}$. The point P_{SVA} will be doubled at iteration $i-1$ only if $k_i = 1$. Several ECSMs are run, with the same scalar and the same base point P. The attacker collects the power consumption trace of the doubling at iteration $i-1$. If $k_i = 1$, P_{SVA} will be doubled at iteration $i-1$ in each ECSM.

Remark 8.17. Taking advantage of same-values points relative to addition is more difficult. Indeed, the attacker needs to find a base point P such that P and $Q = [c]P$, with $c = (k_{n-1}, \ldots, k_{i+1}, 1)_2$, are same-values point relative to the addition used. Finding such a point P is currently difficult if c is large. This issue is discussed in [IT03] and [AT03] for similar reasons.

Detecting the same-values point. To detect if P_{SVA} occurs each time, one can apply the method described in Section 8.2.3.1 for detecting the same values.

Remark 8.18. The same-values point is chosen such that the same values are used in the same field operation (multiplication or addition) and in the same side (left or right). This makes it possible to perform a collision analysis without any synchronization procedure. The attack is still possible if it is not the case, but a strong study on the field arithmetic module is required for a synchronization because the sensitive data is not manipulated at the same time within the field operation.

Compared with the RSCA and the ZSCA, the number of possible same-values points on a curve is very large. This makes it very hard to find a curve that does not contain any same-values points. We do not know if it is even possible.

As opposed to the RSCA and the ZSCA, several traces are necessary to detect if the particular point occurs. Therefore, scalar randomization techniques thwart this attack.

Like the RSCA and the ZSCA, the Point Blinding (see Section 8.3.5) thwarts the attack since the point $P_{\text{SVA}} + R$ with an unknown pseudo-random point R will occur instead of P_{SVA}.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: must contain same-values points,
- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: none,
- **Number of Executions Needed**: multiple,
- **Input Access**: chosen,
- **Output Access**: unnecessary.

8.14 Horizontal SVA

We propose to extend the SVA which has been introduced as a vertical attack in our paper [MGD+12]. We show that, in some conditions, the occurrence of the particular point can be detected with a single trace.

Stronger Conditions of Same-Values Points. In the vertical SVA described in the previous section, the attacker takes advantage of points that have same values occurring in an elliptic curve formula. Here, we take advantage of points with stronger conditions. In addition of the occurrence of the same values, those same values will be inputs of the modular square operation. We can compare the power consumption of the squares to detect if they have indeed the same input. Among all conditions of Theorems 8.12 and 8.14, the conditions below give the required result:

1. $x_1 = 1$: this condition implies that the inputs of the squares at lines 5 ($X_1^2 = (x_1 Z_1^2)^2$) and 11 ($(Z_1^2)^2$) of Algorithm 39 are the same.

2. $2y_1 = 3x_1^2 + a$: this condition implies that the input of the square at line 14 ($(3X_1^2 + aZ_1^4)^2 = ((3x_1^2 + a)Z_1^4)^2$) of Algorithm 39 and the input of the square of the value $Z_3 = 2Y_1Z_1 = 2y_1Z_1^4$, which will occur during the addition or the doubling of P_3 at the next iteration of the ECSM, are the same.

3. $y_2 - y_1 = x_2 - x_1$: this condition implies that the input of the square at line 16 ($F^2 = (Y_2Z_1^3 - Y_1Z_2^3)^2 = ((y_2 - y_1)Z_1^3Z_2^3)^2$ and the input of the square of the value $Z_3 = Z_1Z_2E = ((X_2Z_1^2 - Y_2Z_2^2)Z_1Z_2)^2 = ((x_2 - x_1)Z_1^3Z_2^3)^2$ of Algorithm 37, which will occur during the addition or doubling of P_3 at the next iteration of the ECSM, are the same.

The attacker analyses the trace segments of the two squares to determine if the value squared is the same and conclude on the current bit.

Detecting the Same Inputs in the Squares. If the noise signal is low, a simple difference of the trace segments is enough to detect if the same value is manipulated. If the difference is near zero, the inputs of the two multiplications are equal. This is illustrated in Figure 8.3. If the noise signal is high, a more sophisticated tool can be used, such as the Euclidean distance or the correlation with the points of interest of the traces. The trace segments of the squares to compare can be seen as random variables X, Y. The construction of such random variables are illustrated in Figure 8.4.

The number of possible particular points is reduced compared with the classical SVA of the previous section (two conditions instead of thirty-three for ECDBL). Finding a curve that does not contain any same-values points with the strong condition is feasible. Moreover, for the three conditions listed above, the Random Curve Isomorphism (see Section 8.3.7) thwarts the attack. This information has to be taken with great caution because it is not necessarily the case for other formulæ.

Compared with the vertical version, a single trace is enough to detect if the particular point appears. However, the attack is recursive and n executions of the ECSM is required to reveal the whole scalar. Therefore, like the RSCA and the ZSCA, scalar randomization techniques help prevent the horizontal SVA since the recursive process is broken. However, an attacker can target several bits at a time by guessing several bits instead of only one. Several bits of the randomized scalar can be recovered and reveal some information of the initial scalar.

Like the RSCA, the ZSCA and the classical SVA, the Point Blinding described in Section 8.3.5 thwarts the attack.

We introduced a new kind of attacks and proved the danger of repetition of same values. This was in fact the first Same-Values Analysis that we published. Later, we tried to extend this attack and take advantage of repetition of values on other implementations and proposed the other SVA already described in Sections 8.2.2.2, 8.2.3.1 and 8.2.3.2.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: the curve must contain same-value points which brings same values that will be squared,
- **Implementation Access**: knowledge of the ECSM and the elliptic curve formulæ,
- **Implementation Specificity**: none,
- **Number of Executions Needed**: n,
- **Input Access**: chosen,
- **Output Access**: unnecessary.

8.15 Particular Point Timing Attack [SST04]

Sato, Schepers and Takagi introduced another timing attack. Like the classical Timing Attack described in Section 8.1, this attack takes advantage of the conditional final reduction of the Montgomery multiplication. It works only on curves with parameter $a = -3$. Such curves enable computing the variable C of ECDBL (Algorithm 11) as follows (see Remark 3.1):

$$C = 3(X_1 + Z_1^2)(X_1 - Z_1^2) \tag{8.1}$$

The attack exploits the occurrence of a special point: $P = (2, y)$. In Jacobian coordinates, $P = (2Z_1^2, yZ_1^3, Z_1)$ for some $Z_1 \in \mathbb{F}_p^*$. When P is doubled, substitute its coordinates into Equation (8.1) leads to:

$$C \leftarrow 3(3Z_1^2)(Z_1^2)$$

In [SST04], the authors show that the probability of the final reduction during the Montgomery multiplication of α, β is higher if $\beta = 3\alpha$ than for random values.

If the attacker judiciously chooses the base point, this point occurs only on a certain hypothesis of the scalar. In this case, the average timing of the ECSMs is higher than random inputs.

The particular point is not well randomized by the Random Projective Coordinates countermeasure (see Section 8.3.6). Indeed, whatever the value of Z_1, the inputs of the modular multiplication when computing C will still be $\alpha, \beta = 3\alpha$ for some $\alpha \in \mathbb{F}_p$. The attack can therefore be applied even if this countermeasure is present.

In the following, we show that the Random Curve Isomorphism countermeasure (see Section 8.3.7) thwarts the attack. Indeed, if the points are randomized with the isomorphism defined

as $\varphi\colon (x,y) \to (u^{-2}x, u^{-3}y)$ for some random $u \in F_p^*$, the special point $P = (2Z_1^2, yZ_1^3, Z_1)$ is randomized as $\varphi(P) = P' = (2u^{-2}Z_1^2, yu^{-3}Z_1^3, Z_1)$ in Jacobian coordinates. Substitute its coordinates into Equation (8.1):

$$C \leftarrow 3(Z_1^2(2u^{-2}+1))(Z_1^2(2u^{-2}-1))$$

The inputs of the modular multiplication: $\alpha = (Z_1^2(2u^{-2}+1)), \beta = (Z_1^2(2u^{-2}-1))$ do not verify $\beta = 3\alpha$ if $u \neq \pm 1$.

A more drastic method is to use the Constant Time of Field Operations countermeasure described in Section 8.1.1.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: elliptic curve parameter $a = -3$, must contain a point of the form $(2, y)$ for some $y \in \mathbb{F}_p$,
- **Implementation Access**: full knowledge of all algorithms,
- **Implementation Specificity**: deterministic and non-constant time execution of modular multiplications, fast doubling,
- **Number of Executions Needed**: multiple,
- **Input Access**: chosen,
- **Output Access**: unnecessary.

8.16 Invalid Curve Attack [CJ05]

This weak curve attack is similar to the Invalid Point Attack described in Section 8.5. Rather than injecting a fault on the base point P, Ciet and Joye propose to inject a fault on the curve parameters, particularly on the parameter a of the curve.

Denote by $\tilde{a}$ the corrupted value. Since the parameter b of the initial curve is generally not used, $P = (x_P, y_P)$ lies on another curve $\tilde{E}\colon y^2 = x^3 + \tilde{a}x + \tilde{b}$.

The ECSM is performed on the weak curve E'. The attacker recovers the result $\tilde{Q} = (x_{\tilde{Q}}, y_{\tilde{Q}}) = [k]P$. The parameters $\tilde{a}, \tilde{b}$ of the curve $\tilde{E}$ can be retrieved from the following equations system:

$$\begin{cases} \tilde{a}x_P + \tilde{b} = y_P^2 - x_P^3 \\ \tilde{a}x_{\tilde{Q}} + \tilde{b} = y_{\tilde{Q}}^2 - x_{\tilde{Q}}^3 \end{cases}$$

The attacker can solve the ECDLP on the weak curve $\tilde{E}$ to recover $k \bmod ord(P)$.

Attack Context:

- **Key recovery**: each time, an ECDLP on $\tilde{E}$ is performed to recover $k \bmod ord(P)$; the full key is then recovered using the CRT,
- **Elliptic Curve Specificity**: none,

- **Implementation Access**: knowledge of the location of the curve parameters' memory blocks, knowledge of the ECSM,
- **Implementation Specificity**: same addresses of the curve parameters on different executions,
- **Number of Executions Needed**: less than n (it depends on the order of P on the weak curve $\tilde{E}$),
- **Input Access**: known,
- **Output Access**: known,
- **Fault Model**: data randomization on a single memory block of size n.

8.16.1 Curve Integrity Check [CJ05]

Ciet and Joye pointed out the necessity to verify that the public parameters were not disturbed. A cyclic redundancy check is performed on the curve parameters to verify that no fault was introduced. For a full protection, we must check all curve parameters: a, b and the modulus p.

Cost: 3 CRC_n

8.17 Sign Change Fault Attack [BOS06]

Blömer, Otto and Seifert introduced a DFA. Their idea is to inject a fault to change the sign of an intermediate point during the ECSM. Let's take for example the Double-and-Add always method (Algorithm 19).

First, a correct result $Q = [k]P$ is recovered from a first ECSM. A second ECSM is run with the same scalar and the same input. At iteration i, a fault is induced during the addition of points to switch the sign of R_0. The operation $R_{1-k_i} \leftarrow R_0 + P$ becomes $R_{1-k_i} \leftarrow P - R_0$. If $k_i = 0$, the result is correct, and the attacker tries again at another iteration. If $k_i = 1$, the incorrect result $\tilde{Q}$ is equal to $\tilde{Q} = [(k_i, \ldots, k_0)_2]P - [2^i(k_{n-1}, \ldots, k_{i+1})_2]P$.

The attacker computes $Q + \tilde{Q} = [2(k_i, \ldots, k_0)_2]P$. If i is small enough, the attacker can perform the ECDLP on $Q + \tilde{Q}$ to recover $(k_i, \ldots, k_0)_2$ with a complexity of $2^{i/2}$ using the baby-step giant-step method [Sha71].

The method is iterated to recover the other bits.

The inverse of point $R = (X, Y, Z)$ is $-R = (X, -Y, Z)$. To change the sign of a point, the control signal of the two's complement of the operand is switched when loading Y [BOS06]. One can also inject a fault on an opcode during the elliptic curve addition formula to switch from a field addition to a field subtraction.

Since all points still lie on the given elliptic curve, the Output Point Validity and the Curve Integrity Check do not detect the fault (see Sections 8.5.1 and 8.16.1 respectively).

Attack Context:

- **Key recovery**: recursive, several ECDLPs are performed,
- **Elliptic Curve Specificity**: none,

- **Implementation Access**: full knowledge of all algorithms, knowledge of the architecture of the modular arithmetic operator,
- **Implementation Specificity**: same addresses of the coordinates' intermediate point on different executions,
- **Number of Executions Needed**: less than n,
- **Input Access**: known and constant,
- **Output Access**: known,
- **Fault Model**: data randomization on a single control signal or modifying opcode.

8.18 Coherence Check [Gir06]

This countermeasure was introduced by Giraud to protect RSA implementations against fault attacks. It can be adapted for ECC. A verification is performed at the end of to the Montgomery Ladder (Algorithm 21) to check the integrity of the result.

At the end of each iteration, R_0 and R_1 verify $R_1 - R_0 = P$. One can verify at the end of the ECSM that the equality stands. Any fault on the curve parameters or the intermediate points *after* the initialization phase (lines 1 and 2 of algorithm 21) will be detected with very high probability [DH11].

This countermeasure can be adapted for Algorithm 20. Indeed, at the end of each iteration, $R_0 + R_1 + P = [(k_i, \ldots, k_0)_2]P + [(\bar{k}_i, \ldots, \bar{k}_0)_2]P + P = [2^{i+1}]P = R_2$. The equality can be verified at the end.

The countermeasure is also applicable on Algorithm 25 [BHT09]. The equality is similar to Algorithm 20: $R_0 + R_1 + P = R_2$ is verified at each iteration.

In addition of preventing the Weak Curve attacks (Sections 8.6, 8.5 and 8.16), it also brings security against the Sign Change Fault attack (Section 8.17). This also thwarts the C Safe-Error (Section 8.2.1.1) since the dummy operations introduced are in fact used at the end for the check.

The countermeasure costs one or two elliptic curve additions (depending on the algorithm), the field operations necessary to compare two points in Jacobian coordinates (a conversion to the same Z coordinate is required) and two memory blocks needed to store the base point P for the final check.

Cost: 1 or 2 ECADD_n, 2 mSQR_n, 6 mMUL_n, 2 mADD_n, 2 MEM_n

8.19 Zero Word and SSCA [AVFM07]

Amiel, Villegas, Feix and Marcel suggest to combine a fault attack and a SSCA.

Let's take the example of the Left-to-Right sliding window NAF method (Algorithm 16). At the beginning of the ECSM, a fault is introduced to set a word of one coordinate of one of the read-only points (for example P_1) to zero. A multiplication with an operand with a zero word is easily detected by observing the trace (see Figure 8.17).

This attack permits to recover all bits of the scalar with a single trace. This attack works on ECSMs in which at least one point is read only, and its usage depends on the scalar. This is the case for the Left-to-Right window method (Algorithm 16), the Shamir's trick (Algorithms

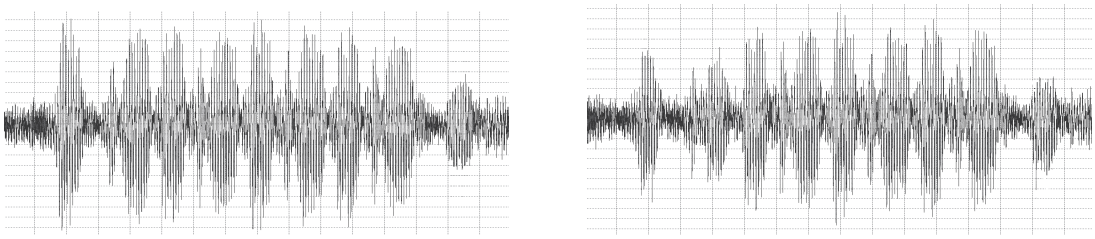


Figure 8.17: Power consumption of a random Montgomery multiplication (left curve) and a Montgomery multiplication with the first word of the first operand being zero (right curve)

18 and 23), and the BRIP (Algorithm 24).

Checking the validity of the output is ineffective since the scalar is deduced from the side-channel observation.

Attack Context:

- **Key recovery**: independent bits,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the ECSM, knowledge of the location of the memory blocks,
- **Implementation Specificity**: ECSM where a point is read only,
- **Number of Executions Needed**: 1,
- **Input Access**: unnecessary,
- **Output Access**: unnecessary,
- **Fault Model**: resetting data on a word.

8.20 Template Attack on ECDSA [MO08]

Medwed and Oswald could mount a template attack on ECDSA to recover a few bits of several different scalars from different signatures.

The number of bits was large enough to recover the static private key of the signature using the lattice attack described in [HS01].

There is no theoretical obstruction which prevents this attack from recovering all bits of the scalar. Indeed, when the first bits are known, the attacker can construct new templates using her own controllable device with the known first bits and the next unknown few bits. The templates are compared with the same trace of the targeted device. However, no experimental results have been proposed to recover the whole scalar.

Randomizing the base point using the Random Projective Coordinates (Section 8.3.6), the Point Blinding (Section 8.3.5) or the Random Curve Isomorphism (Section 8.3.7) thwarts the attack since the stage of constructing the templates cannot be done.

Attack Context:

- **Key recovery**: recursive,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: access to exactly the same controllable chip as the target,
- **Implementation Specificity**: none,
- **Number of Executions Needed**: 1,
- **Input Access**: known,
- **Output Access**: unnecessary.

8.21 Twist Curve Attack [FLRV08]

If the Montgomery Ladder is used, addition and doubling formulæ can be performed without the y coordinate [IMT02]. Fouque, Lercier, Réal and Valette suggest to inject a fault on the x coordinate of the base point $P = (x_P, y_P)$ lying on the elliptic curve E: $y^2 = x^3 + ax + b$.

Let $\tilde{P} = (x_{\tilde{P}}, y_{\tilde{P}})$ denotes the faulted point. $y_{\tilde{P}}$ is unused during the computation of $[k]\tilde{P}$. Two possibilities arise:

- $x_{\tilde{P}}^3 + ax_{\tilde{P}} + b$ is a square. In this case, $y_{\tilde{P}}$ is a solution of the equation $y^2 = x_{\tilde{P}}^3 + ax_{\tilde{P}} + b$ in $\mathbb{F}_p$ and $\tilde{P}$ lies on the curve E. The x coordinate of the result point $\tilde{Q} = [k]\tilde{P}$ gives nothing.
- $x_{\tilde{P}}^3 + ax_{\tilde{P}} + b$ is not a square. In this case, $y_{\tilde{P}}$ is a solution of the equation $y^2 = x_{\tilde{P}}^3 + ax_{\tilde{P}} + b$ in $\mathbb{F}_{p^2}$ and $\tilde{P}$ lies on the twist curve $\tilde{E}$ defined over $\mathbb{F}_{p^2}$. If $\tilde{E}$ is weak, the ECDLP is feasible.

In the latter case, one method to recover the faulted value $x_{\tilde{P}} = x_P \oplus \varepsilon$ is to try all possible values of ε and compute the ECDLP with $\tilde{P}$ and the result point $\tilde{Q} = [k]P$. Therefore, the number of bits affected by the fault has to be small (max 8 or 16)[13].

The output point validity described in Section 8.5.1 thwarts the attack *only* if a full check is performed, *i.e.* with all curve parameters and all coordinates. For example, it is not enough to verify if $x_Q^3 + ax_Q + b$ is a square, x_Q being the x coordinate of the ECSM's output. A second fault can be induced at the end to bypass the check with probability 1/2 [FLRV08].

Attack Context:

- **Key recovery**: performing 2^8 or 2^{16} ECDLPs on $\tilde{E}$ (depending on the precision of the fault),
- **Elliptic Curve Specificity**: the twist of the curve is weak,
- **Implementation Access**: knowledge of the location of the base point's memory blocks,
- **Implementation Specificity**: elliptic curve formulæ without the y coordinate, same addresses of the coordinates' base point on different executions,

[13] Another method to find the value $x_{\tilde{P}}$, with less computational effort, is described in [FLRV08]. However, it needs more faulted results.

- **Number of Executions Needed**: 1,
- **Input Access**: known,
- **Output Access**: known,
- **Fault Model**: data randomization on a single register of small size (maximum 8 or 16).

8.22 Invalid Point Attack and SSCA [FGV11]

Fan, Gierliches and Vercauteren propose to combine an Invalid Point Attack (see Section 8.5) and a SSCA.

At the beginning of the ECSM, a known fault is introduced to the base point P. The faulty point $\tilde{P}$ lies on another curve, and has a very low order $ord(\tilde{P})$. Eventually, the point at infinity $\mathcal{O}$ will occur during the computation of $[k]\tilde{P}$. In embedded systems, the point at infinity is generally not managed for efficiency reasons. This is understandable since the point at infinity should not appear in a normal mode of operation. The elliptic curve points operations will then be incorrect and zero values will appear and remain till the end of the ECSM. The manipulation of a zero value is easily detected by SSCA (see Figure 8.15). The number of iterations before the apparition of the point at infinity reveals k mod $ord(\tilde{P})$, or the most significant bits of k modulo $ord(\tilde{P})$ in the case of a Left-to-Right ECSM.

Unlike the classical Invalid Point Attack (see Section 8.5), the attacker does not need the output point: the attack works even if a validity check is performed at the end of the ECSM (see Sections 8.5.1 and 8.18).

The choice of the base point is made by the attacker because it has to be a *neighbour* of a point of low order $\tilde{P}$. P and $\tilde{P}$ are *neighbours* in the sense that only one bit differs between P and $\tilde{P}$ [FGV11].

The Point Blinding of Section 8.3.5 and any scalar randomization prevent the attack [FGV11].

Attack Context:

- **Key recovery**: each time, k mod $ord(\tilde{P})$ is revealed; the full key is then recovered using the CRT,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the base point's memory blocks, knowledge of the ECSM,
- **Implementation Specificity**: same addresses of the coordinates' base point on different executions,
- **Number of Executions Needed**: less than n (it depends on the order of $\tilde{P}$),
- **Input Access**: chosen,
- **Output Access**: unnecessary,
- **Fault Model**: data randomization on a single bit.

8.22.1 Input Point Validity after a Randomization [FGV11]

Checking if the base point lies on the given elliptic curve is generally done at the cryptographic level. However, a fault can be induced after the verification.

Performing the check *after* a randomization of the input, such as the Random Projective Coordinates (Section 8.3.6), the Random Curve Isomorphism (Section 8.3.7) or the Point Blinding (Section 8.3.5), is very powerful. Indeed, the fault injected after the check will necessarily be random, since all values are randomized.

Cost: 5 mMUL_n, 4 mSQR_n, 4 mADD_n

8.23 Fault Attack on Coordinates Conversion [MMNT13]

We present our new DFA. In the previous fault attacks, the fault is either introduced at the beginning (Weak Curve attacks) or during the ECSM (Safe-Errors and the other DFAs). Here, the fault is introduced at the very end of the ECSM, during the projective to affine coordinates conversion. This enables to retrieve the projective coordinates of the result of the ECSM $[k]P$. Naccache, Smart and Stern showed that when the result is given in projective coordinates, an attacker can recover information on k [NSS04].

Leakage in Projective Coordinates. First, we briefly overview the attack described in [NSS04] when the attacker has access to the Jacobian coordinates of the output.

Assume that Algorithm 14 is used to compute $Q = [k]P$ with mixed coordinates (P is in affine coordinates).

We denote by $A_i = (X_i, Y_i, Z_i)$ the value of point A at the end of iteration i in Algorithm 14. The attacker knows the output $A_0 = Q = (X_0, Y_0, Z_0)$ in Jacobian coordinates and the input $P = (x_P, y_P)$ in affine coordinates. The attacker will attempt to reverse the scalar multiplication process *i.e.* replace doubling by halving and replace additions of P by subtractions.

If $k_0 = 0$, A_1 can be recovered by halving A_0. Given Algorithm 11:

$$Z_0 = 2Y_1Z_1 = 2y_1Z_1^4 \;\Rightarrow\; Z_1^4 = \frac{Z_0}{2y_1}$$

y_1 is obtained by computing $(x_1, y_1) = [2^{-1} \bmod \#E]Q$ in affine coordinates which is always possible. We need to compute a fourth root to obtain Z_1 from Z_0 and y_1:

- if $p \equiv 1 \pmod 4$, then computing a fourth root is possible in a quarter of the cases and yields four values.
- if $p \equiv 3 \pmod 4$, then computing a fourth root is possible for half of the inputs and, when possible, this computation yields two values.

We can easily obtain X_1 and Y_1 from Z_1.

If, on the other hand, $k_0 = 1$, A_1 can be recovered by subtracting P from A and halving. We denote by (X_t, Y_t, Z_t) the intermediate point between doubling (step $A \leftarrow \text{ECDBL}(A)$) and

addition (step if $k_i = 1$ then $A \leftarrow \text{ECADD}(A, P)$). Given Algorithm 10 in mixed coordinates, we have:

$$Z_0 = (x_P Z_t^2 - X_t) Z_t \;\Rightarrow\; Z_t^3 = \frac{Z_0}{x_P - x_t}$$

x_t is obtained by computing $(x_t, y_t) = Q - P$ in affine coordinates. We need to compute a cubic root to obtain Z_t from Z_0, x_p and x_t:

- if $p \equiv 1 \pmod 3$, then extracting a cubic root is possible in a third of the cases and, when possible, this calculation yields one of three possible values.
- if $p \equiv 2 \pmod 3$, then extracting a cubic root is always possible and yields a unique value.

We can easily obtain X_t and Y_t from Z_t. After subtraction, the attacker must halve (X_t, Y_t, Z_t) as described previously:

$$Z_1^4 = \frac{Z_t}{2y_t}.$$

From this observation, the opponent can recover the least significant bit of k. Indeed, if the value $\frac{Z_0}{2y_1}$ isn't a fourth power, the opponent immediately concludes that $k_0 = 1$. If $\frac{Z_0}{2y_1}$ is a fourth power, then the attacker must try the subtraction and halving step. If subtracting P from A_0 or halving A_t is impossible, the attacker concludes that $k_0 = 0$. If both steps are possible (which happens with non-negligible probability), the attacker cannot immediately identify k_0, but can hope to do so by backtracking, *i.e.* guessing the values of k_1, k_2, etc. and computing the corresponding intermediate points until reaching one of the previous contradictions.

Once k_0 is known, the opponent can iterate the procedure starting with k_1 and so forth to extract a few more bits of k. Note that several candidate values for Z_1 arise from the reversal process as the corresponding equations have several roots, and backtracking is usually required to determine the correct one.

[NSS04] reports experimental data on the number of recovered bits and success probabilities. The success highly depends on the value of p mod 12.

To prevent this attack, the defender should in principle output results in affine coordinates. Another possible countermeasure suggested in [NSS04] is to randomize the output, replacing (X_0, Y_0, Z_0) by $(r^2 X_0, r^3 Y_0, r Z_0)$ for some random $r \in \mathbb{F}_p^*$, which effectively avoids any possible leakage from the Jacobian representation.

As a side note, we point out that, while [NSS04] also claims that attacks are thwarted by randomly flipping the sign of Z_0, this is incorrect: just as k_1 can be recovered with significant probability even though Z_1 is only known up to a sign (by simply trying both possibilities and backtracking until a contradiction is reached), k_0 can also be recovered even when Z_0 is only known up to a sign. This observation is important in our case, as the fault attacks described hereafter retrieve Z_0^2 rather than Z_0 itself.

Projective-to-Affine Conversion. The following procedure converts the point $P = (X, Y, Z) = (xZ^2, yZ^3, Z)$ from Jacobian to affine coordinates (x, y).

$$\text{CONVERT}(X, Y, Z) = \left\{ \begin{array}{lcll} r & \leftarrow & Z^{-1} & \\ s & \leftarrow & r^2 & \\ x & \leftarrow & X \cdot s & \\ t & \leftarrow & Y \cdot s & \\ y & \leftarrow & t \cdot r & \text{return}(x, y) \end{array} \right. \tag{8.2}$$

Faults during conversion. In standardized cryptographic protocols based on elliptic curves, the computed points are given in affine coordinates, and hence [NSS04] does not apply. Our idea is to corrupt the conversion process, so that the faulty affine results reveal the missing Z coordinate. Suppose that an error corrupted s just after the step $s \leftarrow r^2$ (of Process (8.2)). The corrupted $s + \varepsilon$ yields:

$$\tilde{x} = X(s + \varepsilon) \Rightarrow \tilde{x} = x + xZ^2\varepsilon \tag{8.3}$$

$$\tilde{y} = Y(s + \varepsilon)r \Rightarrow \tilde{y} = y + yZ^2\varepsilon \tag{8.4}$$

Equations (8.3) and (8.4) imply

$$\frac{\tilde{x}}{x} - 1 = Z^2\varepsilon \tag{8.5}$$

$$\frac{\tilde{y}}{y} - 1 = Z^2\varepsilon \tag{8.6}$$

We will describe three different attacks depending on the fault's precision.

Large Unknown Faults and One Correct Result. Let $\boldsymbol{\varepsilon} = (\varepsilon_1, ..., \varepsilon_l)$ be a vector of l large faults, as illustrated in Figure 8.18. We want to recover $\boldsymbol{\varepsilon}$.

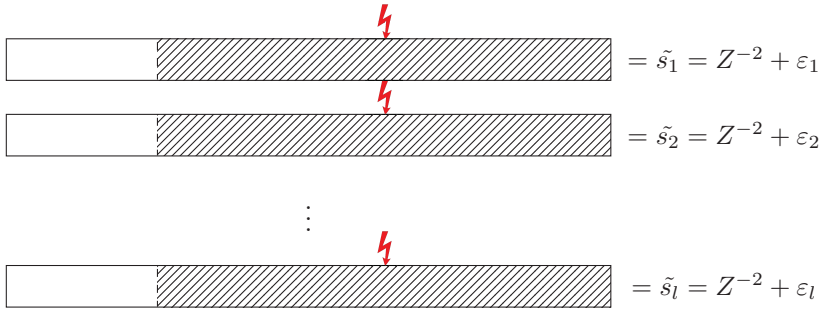


Figure 8.18: Illustration of large fault injections

Each ε_i satisfies an equation of the form (8.5), thus the attacker knows l numbers $u_i = Z^2 \cdot \varepsilon_i \bmod p$ denoted as a vector $\mathbf{u} = (u_1, ..., u_l)$. Assume that $\forall i \in \{1, \ldots, l\}, \varepsilon_i < p^\alpha$ for a number $\alpha < 1$. Let L be the lattice generated by the vector $\mathbf{u}$ and $p\mathbb{Z}^l$ in $\mathbb{Z}^l$ and let $s = Z^{-2} \bmod p$. Since $\boldsymbol{\varepsilon}$ satisfies $\boldsymbol{\varepsilon} = s \cdot \mathbf{u} \bmod p$, $\boldsymbol{\varepsilon}$ is a vector in L, of length $\|\boldsymbol{\varepsilon}\| \lesssim p^\alpha$. Assume further that $g = \gcd(u_1, \ldots, u_l) = 1$. This happens with probability $\approx 1/\zeta(l) \approx 1 - 2^{-l}$, which is very close to 1. Then, we have $\mathrm{vol}(L)^{\frac{1}{l}} = [\mathbb{Z}^l : L]^{\frac{1}{l}} = p^{1-\frac{1}{l}}$. Therefore, we can recover $\boldsymbol{\varepsilon}$ directly by reducing the lattice L using LLL [LLL82] as long as $p^\alpha \ll p^{1-\frac{1}{l}}$, *i.e.* $l > \frac{1}{1-\alpha}$.

The attack can also be carried out when $g > 1$: in that case, LLL will recover $\pm 1/g \cdot \boldsymbol{\varepsilon}$, so exhaustive search on the few possible values of g is enough. However, the probability that $g > 1$ is so small makes this refinement unnecessary.

Size of p (modulus size)	256 bits
Number of errors (l)	9
Error size (percentage of the modulus size)	224 bits (87.5%)
Success probability	99.8%
CPU time	3 ms

Table 8.1: Timings for a SAGE implementation on a 2.27 GHz Intel Core i3 CPU core.

To evaluate the attack, we implemented it in SAGE [SAGE12] (without treating the case $g > 1$) and observed the results given in Table 8.1. The failure rate of $\approx 0.2\%$ corresponds to the cases when $g > 1$, and is consistent with $1/\zeta(9) \approx 0.998$.

Remark 8.19. In the paper [MMNT13], we present an alternative solution where the correct result is not necessary.

Since several faulty results with the same Z coordinate are necessary, any randomization used against the CSCA described in Section 8.3 thwarts this attack. The knowledge of the base point in affine coordinates is necessary for the backtracking algorithm. For this reason, the Point Blinding countermeasure (see Section 8.3.5) is also effective against the attack.

Attack Context:

- **Key recovery**: Naccache's et al.'s backtracking algorithm,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the memory blocks, knowledge of the conversion procedure,
- **Implementation Specificity**: use of Jacobian or projective coordinates,
- **Number of Executions Needed**: ≈ 10 for a few bits,
- **Input Access**: constant,
- **Output Access**: known,
- **Fault Model**: data randomization on a single memory block of size $\approx \frac{4n}{5}$.

Two Faults and a Correct Result. As we have just seen, a correct conversion and two faulty conversions yield the values: $Z^2\varepsilon_1$ and $Z^2\varepsilon_2$ and hence, by modular division $\beta = \varepsilon_1\varepsilon_2^{-1}$. Theorem 8.20 (see [FSW02]) guarantees that ε_1 and ε_2 can be efficiently recovered from β if each ε_i is smaller than the square root of p divided by 2. This problem is known as the *Rational Number Reconstruction* [PW04, WP03] and is typically solved using Gauß' algorithm for finding the shortest vector in a bidimensional lattice [Val91].

Theorem 8.20. *Let $\varepsilon_1, \varepsilon_2 \in \mathbb{Z}$ such that $-A \leq \varepsilon_1 \leq A$ and $0 < \varepsilon_2 \leq B$. Let $p > 2AB$ be a prime and $\beta = \varepsilon_1\varepsilon_2^{-1} \bmod p$. Then $\varepsilon_1, \varepsilon_2$ can be recovered from A, B, β, p in polynomial time.*

Assume that the ε_i are smaller than $\sqrt{p}$, as illustrated in Figure 8.19. Taking $A = B = \lfloor\sqrt{p/2}\rfloor$, we get $2AB < p$. Moreover, $0 \leq \varepsilon_1 \leq A$ and $0 < \varepsilon_2 \leq B$. Thus the attacker can recover ε_1 and ε_2 from β in polynomial time.

If the ε_i are shifted to the left by an arbitrary number of bit positions, this does not change anything as these powers of two will divide out.

The attack is also feasible in the more general unbalanced case when

$$\varepsilon_1\varepsilon_2 \leq p/4. \tag{8.7}$$

In contrast to the case where the ε_i are bound individually (*i.e.* $0 \leq \varepsilon_1 \leq A$ and $0 < \varepsilon_2 \leq B$) we do not have a fixed bound for ε_1 and ε_2 anymore; Equation (8.7) only provides a bound for the *product* $\varepsilon_1\varepsilon_2$. Equation (8.7) implies that there exists $1 \leq i \leq \lfloor n \rfloor$ such that $0 \leq \varepsilon_1 \leq 2^i$ and $0 < \varepsilon_2 \leq p/2^{i+1}$. Then using Theorem 8.20 again, the attacker can recover the pair $(\varepsilon_1, \varepsilon_2)$,

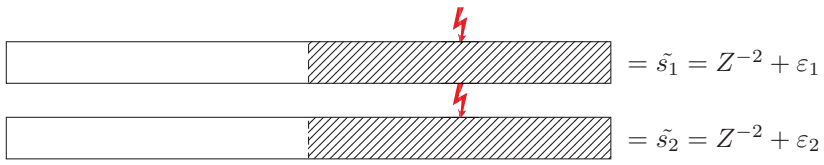


Figure 8.19: Illustration of two fault injections, half the size of p

and hence Z. In principle, there could be several candidate solutions depending on the choice of i, making it necessary to consider many possible values of Z. In practice, however, multiple solutions seem to occur with negligible probability when p is large enough.

Like the previous attack with large faults, any randomization described in Section 8.3 thwarts this attack since two faulty results with the same Z coordinate are necessary. The Point Blinding of Section 8.3.5 is effective as well.

Attack Context:

- **Key recovery**: Naccache's et al's backtracking algorithm,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the memory blocks, knowledge of the conversion procedure,
- **Implementation Specificity**: use of Jacobian or projective coordinates,
- **Number of Executions Needed**: 3 for a few bits,
- **Input Access**: constant,
- **Output Access**: known,
- **Fault Model**: data randomization on a single memory block of size $n/2$.

Known or guessable fault. If ε is known or successfully guessed, then one faulty point $(\tilde{x} = x + xZ^2\varepsilon, \tilde{y} = y + yZ^2\varepsilon)$ and the correct point (x, y) is enough to recover Z (up to the sign).

As opposed to the previous attacks, a single faulty result is necessary for this attack. The security is not guaranteed anymore by the randomization techniques of Section 8.3, except for the Point Blinding because the attacker needs the knowledge of the base point of the ECSM.

Attack Context:

- **Key recovery**: Naccache's et al's backtracking algorithm,
- **Elliptic Curve Specificity**: none,
- **Implementation Access**: knowledge of the location of the memory blocks, knowledge of the conversion procedure,

- **Implementation Specificity**: use of Jacobian or projective coordinates,
- **Number of Executions Needed**: 2 for a few bits,
- **Input Access**: constant,
- **Output Access**: known,
- **Fault Model**: data randomization on a single bit.

Chapter 9

Differential Fault Attacks on ECDSA

Some DFAs previously described rely on the comparison of one erroneous result and the correct result of an ECSM, to recover a few bits of the scalar. That is the case for the classical DFA (Section 8.6), the Sign Change Fault attack (Section 8.17) and the Fault Attack on the Coordinates Conversion with the known fault (Section 8.23).

In ECDSA, the scalar used for the ECSM is randomly chosen for each new signature. In this chapter, we describe a method to attack the ECDSA with the DFAs listed above. This method was initially described in our paper [MMNT13, §6.1].

9.1 Principle of the Method

Let G be the generator of the subgroup considered on the given elliptic curve E, d the secret key the attacker wants to recover, and $P = [d]G$ the corresponding public key.

We suppose that, during the signature procedure of ECDSA (Algorithm 26), a fault is induced during the computation of Q, yielding to the wrong result $\tilde{Q} = (x_{\tilde{Q}}, y_{\tilde{Q}})$. The erroneous signature $(\tilde{r}, \tilde{s})$ satisfies:

$$\begin{aligned}\tilde{r} &= x_{\tilde{Q}} \bmod t \\ \tilde{s} &= k_{inv}(d\tilde{r} + m) \bmod t\end{aligned}$$

From $(\tilde{r}, \tilde{s})$, the attacker can compute:

$$\begin{aligned}\tilde{w} &= \tilde{s}^{-1} && \bmod t \\ \tilde{u}_1 &= \tilde{w} \cdot m && \bmod t \\ \tilde{u}_2 &= \tilde{w} \cdot \tilde{r} && \bmod t \\ R &= [\tilde{u}_1]G + [\tilde{u}_2]P = \left[\tfrac{km}{d\tilde{r}+m}\right] G + \left[\tfrac{k\tilde{r}}{d\tilde{r}+m}\right] P = \left[\tfrac{km}{d\tilde{r}+m}\right] G + \left[\tfrac{dk\tilde{r}}{d\tilde{r}+m}\right] G \\ &= \left[k \cdot \tfrac{d\tilde{r}+m}{d\tilde{r}+m}\right] G = [k]G\end{aligned}$$

The point R is hence the correct value of $Q = (x_Q, y_Q) = [k]G$ that was intended to be computed during the signature. The attacker gets x_Q, y_Q and $x_{\tilde{Q}} \bmod t$. Since $\log_2(p) \approx \log_2(t)$ in most standardized curves, $x_{\tilde{Q}}$ can be guessed from $x_{\tilde{Q}} \bmod t$.

9.2 Attacking ECDSA with the Classical Differential Fault Attack

We recall that for the classical DFA, described in Section 8.6, a fault is induced on an intermediate point of the ECSM. Using the example of the Right-to-Left Double-and-Add method (Algorithm 15), the attacker injects a fault to alter the intermediate point $Q_i = [(k_{i-1}, \ldots, k_0)_2]P$ into $\tilde{Q}_i$ (we use the same notations as in Section 8.6). If the fault is known or can be guessed[1], the attacker can generate $\tilde{Q}$ from $\tilde{Q}_i$ and all possible values of $(k_{i-1}, \ldots, k_0)_2$. The good hypothesis is the one where the x coordinate of $\tilde{Q}$ matches $x_{\tilde{Q}}$.

9.3 Attacking ECDSA with the Sign Change Fault Attack

We recall that, in the attack described in Section 8.17, the sign of an intermediate point at iteration i is switched during the ECSM. The attacker gets the x coordinate of the ECSM's result $\tilde{Q}$ from the signature. The correct result Q is retrieved using the method described above. Since $\tilde{Q}$ lies on the curve, the attacker can recover the missing y coordinates with probability $1/2$ using the curve equation. She then computes $Q + \tilde{Q} = [2(k_i, \ldots, k_0)_2]P$ and preforms an ECDLP to recover $(k_i, \ldots, k_0)_2$. If she is not successful, her guess on y was wrong and tries the other possibility.

9.4 Attacking ECDSA with the Fault on the Coordinates Conversion

In our attack described in Section 8.23, a fault is induced during the affine to projective coordinates conversion process. The attacker gets the faulty x coordinate of the result $x_{\tilde{Q}} = x_Q + x_Q Z_Q^2 \varepsilon$ with x_Q, Z_Q unknown. $x_{\tilde{Q}}, \varepsilon$ (the fault targets only a bit) are known. The attacker can recover x_Q using the method previously described. Z_Q is then retrieved and this makes the attack described in Section 8.23 and in [NSS04] possible. A few bits of k can be recovered.

9.5 Synthesis

We showed how to use the properties of ECDSA to compare a correct and an erroneous result of an ECSM. For the three DFAs, a few bits of the ephemeral scalar k can be recovered. Iterating the procedure can reveal a few bits of several ephemeral scalars. This is precisely the scenario considered in [HS01] allowing recover the private key d.

Our analysis showed that the ECDSA is not naturally immune to attacks where several ECSMs have to be run with the same scalar.

[1] For example if a single bit is switched but the attacker does not know which one, she can try all bits

Chapter 10

Summary of the Context of the Attacks

In this chapter, we give a synthesis of the attacks depending on the context. This clear synthesis is useful when implementing an embedded ECC application. Depending on the protocols intended to support or on the targeted implementation, some attacks are not feasible and a protection against this attack is not necessary.

10.1 Key Recovery

The synthesis of the key recovery process is given in Table 10.1. This makes an understanding of the attacks. We can anticipate on some future refinements of the attacks since the key recovery process does not evolve drastically.

10.2 Elliptic Curve Specificity

Few attacks work on specific curves only. We give the list below.

- **Particular Point Timing Attack (Section 8.15)**: the parameter a of the curve is equal to -3 and the curve contains a point of the form $(2, y)$ for some $y \in \mathbb{F}_p$.
- **RSCA (Section 8.11)**: the curve contains a point of the form $(0, y)$ for some $y \in \mathbb{F}_p$.
- **ZSCA (Section 8.12)**: the curve contains at least one zero-value point.
- **SVA (Section 8.13)**: the curve contains at least one same-values point.
- **Horizontal SVA (Section 8.14)**: the curve contains at least one same-values point where the same values will be squared.
- **Twist Curve Attack (Section 8.21)**: the twist of the curve is weak.

This synthesis is useful if the application has to support only a few known curves. We can easily check if the curves verify the condition for each attack. If it is not the case, a protection against the corresponding attack is not necessary.

	Independent Bits	Recursive	ECDLP with smaller scalars	ECBLP on a weak curve	Chinese Remainder Theorem	Naccaches et al's backtracking algorithm
Classical Timing Attack Section 8.1		✓				
Particular Point Timing Attack Section 8.15		✓				
Classical SSCA Section 8.2		✓				
SSCA on unified formulæ Section 8.2.2.1		✓				
RSCA Section 8.11		✓				
ZSCA Section 8.12		✓				
Template Attack Section 8.20		✓				
CSCA Section 8.3		✓				
Address-bit DSCA Section 8.9	✓					
Doubling Attack Section 8.10	✓					
Carry Leakage Attack Section 8.3.1.1	✓					
SVA Section 8.13		✓				
SVA on Atomicity Section 8.2.3.1		✓				
Big Mac Section 8.7	✓					
Horizontal Correlation SCA Section 8.8		✓				
Horizontal SVA Section 8.14		✓				
Horizontal SVA on Unified Formulæ Section 8.2.2.2		✓				
Horizontal SVA on Atomicity Section 8.2.3.2		✓				
C Safe-Error Section 8.2.1.1	✓					
M Safe-Error Section 8.4	✓					
Invalid Point Attack Section 8.5				✓	✓	
Invalid Curve Attack Section 8.16				✓	✓	
Twist Curve Attack Section 8.21				✓		
Classical DFA Section 8.6		✓				
Sign Change Fault Section 8.17		✓	✓			
Fault Attack on Coordinates Conversion Section 8.23						✓
Combined Attack on Additive Splitting Section 8.3.2.2	✓					
Zero word and SSCA Section 8.19	✓					
Invalid Point Attack and SSCA Section 8.22		✓			✓	

Table 10.1: Synthesis of the Key Recovery Process for each attack

10.3 Implementation Access

The synthesis of the implementation access, *i.e.* the attacker's level of knowledge of the implementation, is given in Table 10.2. This synthesis reveals what the attacker needs to know or guess to succeed.

10.4 Implementation Specificity

The synthesis of the implementation specificity is given in Table 10.3. When implementing an embedded ECC application, it is fast to check if some attacks are not feasible because of the choice of the implementation. Instead of selecting countermeasures to prevent some attacks, one may prefer to implement an embedded ECC that is naturally immune against some attacks.

10.5 Number of Executions Needed

The synthesis of the number of executions needed is given in Table 10.4. For all attacks, the number given corresponds to the number required to recover the whole scalar, except for the fault attacks on the conversion coordinates which permits to recover only a few bits (see Section 8.23).

10.6 Input Access

The synthesis of the input access is given in Table 10.5. When the attacker needs to choose the base point, the attack is not feasible for some protocols such as the ECDSA Signature (Section 5.1) where the input is fixed and constant.

10.7 Output Access

The synthesis of the output access is given in Table 10.6. When the output is needed, the attack is not feasible for some protocols such as the EC-ELGAMAL Decryption (Section 5.3) where the output is intended to be kept inside the embedded system.

10.8 Fault Model

The synthesis of the implementation specificity is given in Table 10.7. This reveals the difficulty of some attacks to mount in practice depending on the accuracy.

	ECSM	Elliptic Curve Formulæ	Modular Arithmetic Algorithms	Word Size of Integers	Modular Arithmetic Module Architecture	Memory Location	Access to the same Controllable Device	Splitting Method	Conversion Procedure
Classical Timing Attack Section 8.1	✓	✓	✓						
Particular Point Timing Attack Section 8.15	✓	✓	✓						
Classical SSCA Section 8.2	✓	✓							
SSCA on unified formulæ Section 8.2.2.1	✓	✓	✓						
RSCA Section 8.11	✓	✓							
ZSCA Section 8.12	✓	✓	✓						
Template Attack Section 8.20							✓		
CSCA Section 8.3	✓	✓	✓	✓					
Address-bit DSCA Section 8.9	✓								
Doubling Attack Section 8.10	✓	✓							
Carry Leakage Attack Section 8.3.1.1				✓				✓	
SVA Section 8.13	✓	✓							
SVA on Atomicity Section 8.2.3.1	✓	✓							
Big Mac Section 8.7	✓	✓	✓	✓	✓				
Horizontal Correlation SCA Section 8.8	✓	✓	✓	✓	✓				
Horizontal SVA Section 8.14	✓	✓	✓						
Horizontal SVA on Unified Formulæ Section 8.2.2.2	✓	✓	✓						
Horizontal SVA on Atomicity Section 8.2.3.2	✓	✓	✓	✓	✓				
C Safe-Error Section 8.2.1.1	✓	✓				✓			
M Safe-Error Section 8.4	✓	✓				✓			
Invalid Point Attack Section 8.5	✓					✓			
Invalid Curve Attack Section 8.16	✓					✓			
Twist Curve Attack Section 8.21						✓			
Classical DFA Section 8.6						✓			
Sign Change Fault Section 8.17	✓	✓			✓	✓			
Fault Attack on Coordinates Conversion Section 8.23						✓			✓
Combined Attack on Additive Splitting Section 8.3.2.2	✓					✓			
Zero word and SSCA Section 8.19		✓						✓	
Invalid Point Attack and SSCA Section 8.22	✓	✓				✓			

Table 10.2: Synthesis of the Implementation Access Context

	Deterministic and Non-Constant Modular Arithmetic	Fast Doubling	Unregular ECSM	Read Only Point	Dummy Operation	Different Addition and Doubling Formulæ	Indistinguishable Formulæ	without y coordinate	Affine Coordinates	Projective Coordinates	Atomicity	Same Addresses	Group Scalar Randomization	Additive Splitting	Wordwise Modular Multiplication
Classical Timing Attack Section 8.1	✓														
Particular Point Timing Attack Section 8.15	✓	✓													
Classical SSCA Section 8.2			✓			✓									
SSCA on unified formulæ Section 8.2.2.1			✓				✓								
Address-bit DSCA Section 8.9												✓			
Doubling Attack Section 8.10									✓						
Carry Leakage Attack Section 8.3.1.1													✓[1]	✓[7]	
SVA on Atomicity Section 8.2.3.1			✓								✓				
Big Mac Section 8.7															✓
Horizontal Correlation SCA Section 8.8															✓
Horizontal SVA on Unified Formulæ Section 8.2.2.2							✓								
Horizontal SVA on Atomicity Section 8.2.3.2											✓				
C Safe-Error Section 8.2.1.1					✓							✓			
M Safe-Error Section 8.4												✓			
Invalid Point Attack Section 8.5												✓			
Invalid Curve Attack Section 8.16												✓			
Twist Curve Attack Section 8.21								✓				✓			
Classical DFA Section 8.6									✓			✓			
Sign Change Fault Section 8.17												✓			
Fault Attack on Coordinates Conversion Section 8.23										✓		✓			
Combined Attack on Additive Splitting Section 8.3.2.2												✓		✓	
Zero word and SSCA Section 8.19				✓								✓			
Invalid Point Attack and SSCA Section 8.22												✓			

Table 10.3: Synthesis of the Implementation Specificity for each attack

	1	2	3	≈ 10	n	Multiple
Classical Timing Attack Section 8.1						✓
Particular Point Timing Attack Section 8.15						✓
Classical SSCA Section 8.2	✓					
SSCA on unified formulæ Section 8.2.2.1	✓					
RSCA Section 8.11					✓	
ZSCA Section 8.12					✓	
Template Attack Section 8.20	✓					
CSCA Section 8.3						✓
Address-bit DSCA Section 8.9						✓
Doubling Attack Section 8.10		✓				
Carry Leakage Attack Section 8.3.1.1						✓
SVA Section 8.13						✓
SVA on Atomicity Section 8.2.3.1						✓
Big Mac Section 8.7	✓					
Horizontal Correlation SCA Section 8.8	✓					
Horizontal SVA Section 8.14					✓	
Horizontal SVA on Unified Formulæ Section 8.2.2.2	✓					
Horizontal SVA on Atomicity Section 8.2.3.2	✓					
C Safe-Error Section 8.2.1.1					✓	
M Safe-Error Section 8.4					✓	
Invalid Point Attack Section 8.5					✓	
Invalid Curve Attack Section 8.16					✓	
Twist Curve Attack Section 8.21	✓					
Classical DFA Section 8.6					✓	
Sign Change Fault Section 8.17					✓	
Fault Attack on Coordinates Conversion (Large Faults) Section 8.23				✓		
Fault Attack on Coordinates Conversion ($n/2$ bit length Faults) Section 8.23			✓			
Fault Attack on Coordinates Conversion (Known Faults) Section 8.23		✓				
Combined Attack on Additive Splitting Section 8.3.2.2						✓
Zero word and SSCA Section 8.19	✓					
Invalid Point Attack and SSCA Section 8.22					✓	

Table 10.4: Synthesis of the Number of Executions Needed for each attack

	Unnecessary	Known	Chosen	Constant	Varying
Classical Timing Attack Section 8.1		✓			✓
Particular Point Timing Attack Section 8.15			✓		✓
Classical SSCA Section 8.2	✓				
SSCA on unified formulæ Section 8.2.2.1	✓				
RSCA Section 8.11			✓		✓
ZSCA Section 8.12			✓		✓
Template Attack Section 8.20		✓			
CSCA Section 8.3		✓			✓
Address-bit DSCA Section 8.9	✓				
Doubling Attack Section 8.10			✓		✓
Carry Leakage Attack Section 8.3.1.1	✓				
SVA Section 8.13			✓		✓
SVA on Atomicity Section 8.2.3.1	✓				
Big Mac Section 8.7	✓				
Horizontal Correlation SCA Section 8.8		✓			
Horizontal SVA Section 8.14			✓		✓
Horizontal SVA on Unified Formulæ Section 8.2.2.2	✓				
Horizontal SVA on Atomicity Section 8.2.3.2	✓				
C Safe-Error Section 8.2.1.1	✓				
M Safe-Error Section 8.4	✓				
Invalid Point Attack Section 8.5		✓			
Invalid Curve Attack Section 8.16		✓			
Twist Curve Attack Section 8.21		✓			
Classical DFA Section 8.6		✓		✓	
Sign Change Fault Section 8.17		✓		✓	
Fault Attack on Coordinates Conversion Section 8.23				✓	
Combined Attack on Additive Splitting Section 8.3.2.2	✓				
Zero word and SSCA Section 8.19	✓				
Invalid Point Attack and SSCA Section 8.22			✓		✓

Table 10.5: Synthesis of the Input Access for each attack

	Unnecessary	Known	Knowledge of the Validity
Classical Timing Attack Section 8.1	✓		
Particular Point Timing Attack Section 8.15	✓		
Classical SSCA Section 8.2	✓		
SSCA on unified formulæ Section 8.2.2.1	✓		
RSCA Section 8.11	✓		
ZSCA Section 8.12	✓		
Template Attack Section 8.20	✓		
CSCA Section 8.3	✓		
Address-bit DSCA Section 8.9	✓		
Doubling Attack Section 8.10	✓		
Carry Leakage Attack Section 8.3.1.1	✓		
SVA Section 8.13	✓		
SVA on Atomicity Section 8.2.3.1	✓		
Big Mac Section 8.7	✓		
Horizontal Correlation SCA Section 8.8	✓		
Horizontal SVA Section 8.14	✓		
Horizontal SVA on Unified Formulæ Section 8.2.2.2	✓		
Horizontal SVA on Atomicity Section 8.2.3.2	✓		
C Safe-Error Section 8.2.1.1			✓
M Safe-Error Section 8.4			✓
Invalid Point Attack Section 8.5		✓	
Invalid Curve Attack Section 8.16		✓	
Twist Curve Attack Section 8.21		✓	
Classical DFA Section 8.6		✓	
Sign Change Fault Section 8.17		✓	
Fault Attack on Coordinates Conversion Section 8.23		✓	
Combined Attack on Additive Splitting Section 8.3.2.2			✓
Zero word and SSCA Section 8.19	✓		
Invalid Point Attack and SSCA Section 8.22	✓		

Table 10.6: Synthesis of the Output Access for each attack

	Data Randomization	Resetting Data	Modifying Opcode	Fault Lentgh (in bits)
C Safe-Error Section 8.2.1.1	✓			size of the arithmetic module
M Safe-Error Section 8.4	✓			n
Invalid Point Attack Section 8.5	✓			n
Invalid Curve Attack Section 8.16	✓			n
Twist Curve Attack Section 8.21	✓			< 16
Classical DFA Section 8.6	✓			1
Sign Change Fault (Switch Control Signal) Section 8.17	✓			size of the control signal of two's complement
Sign Change Fault (Switch Addition to Subtraction) Section 8.17			✓	size of the signal operation command
Fault Attack on Coordinates Conversion (Large Faults) Section 8.23	✓			$\approx \frac{4n}{5}$
Fault Attack on Coordinates Conversion (1/2 bit length Faults) Section 8.23	✓			$n/2$
Fault Attack on Coordinates Conversion (Known Fault) Section 8.23	✓			1
Combined Attack on Additive Splitting (C Safe-Error) Section 8.3.2.2	✓			size of the arithmetic module
Combined Attack on Additive Splitting (M Safe-Error) Section 8.3.2.2	✓			n
Zero word and SSCA Section 8.19		✓		size of a word
Invalid Point Attack and SSCA Section 8.22	✓			1

Table 10.7: Synthesis of the Fault Model for each attack

Chapter 11

Synthesis of the Attacks versus the Countermeasures

In this chapter, we give a synthesis of the efficiency of each countermeasure against the different attacks on ECC. It is displayed of the form of a table (Table 11.1) inspired from [FGD+10]. It was completed with more recent attacks and we were able to fill some of the boxes from our analysis. We use the following symbols:

- ✓ means that the countermeasure thwarts the attack,
- $\sim$ means that the countermeasure highly disturbs the attack but does not guarantee a full protection,
- $\times$ means that the countermeasure brings the specified vulnerability to the implementation,
- $-$ means it has been shown that the countermeasure is ineffective against the attack, despite the apparent link between them,
- an empty cell means that either the countermeasure and the attack are clearly unrelated or there is no concrete published study on the effect of the countermeasure on the attack.

	Passive Attacks																		Active Attacks									Combined Attacks					
	TA		SSCA					Vertical Analysis						Horizontal Analysis					Safe-Error		Weak Curve			DFA			Attacks against Additive Splitting					
	Classical TA	Particular Point TA	Classical SSCA	SSCA on Unified Formulæ	RSCA	ZSCA	Template Attack	Classical CSCA	Address-bit DSCA	Doubling Attack	Carry Leakage	Classical SVA	SVA on Atomicity	Big Mac Attack	Horizontal Correlation SCA	Horizontal SVA	Horizontal SVA on Unified Formulæ	Horizontal SVA on Atomicity	C Safe-Error	M Safe-Error	Invalid Point Attack	Invalid Curve Attack	Twist Curve Attack	Classifcal DFA	Sign Change	Fault on Projective-to-affine Conversion	High Order C Safe-Error	High Order M Safe-Error	C Safe-Error and Address-bit DSCAA	M Safe-Error and Address-bit DSCAA	Zero Word and SSCAA	Point of low Order and SSCAA
Constant Time Arithmetic	✓	✓		✓																												
Regular ECSM			✓	✓															×[a]								×[a]		×[a]		✓[b]	
Unified Formulæ			✓	×													×															
Side-Channel Atomicity			✓										×					×	×								×		×			
Isomorphism Shifting					✓																											
Random Coordinates	✓	-			-	-	✓	✓		✓		-			✓	-								✓		✓[c]						
Random Curve Isomorphism	✓	✓			-	-	✓	✓		✓		-			✓									✓		✓[c]						
Point Blinding	✓	✓			✓	✓	✓	✓		-		✓			✓	✓								✓		✓						✓
Group Scalar Randomization	✓	✓			~	~		✓	✓	-	×	✓				~			✓	✓				✓		✓[c]						✓
Additive Splitting	✓	✓			~	~		✓	✓		×	✓				~			✓	✓				✓		✓[c]	×	×	×	×		✓
Euclidean Splitting	✓	✓			~	~		✓	✓			✓				~			✓	✓				✓		✓[c]						✓
Multiplicative Splitting	✓	✓			~	~		✓	✓	-		✓				~			✓	✓				✓		✓[c]						✓
Random Register Address									✓											✓								✓		✓		
Random Multiplication														✓	✓	✓	✓	✓														
Coherence Check																			✓		✓	✓	✓	✓	✓							
Output Point Validity																					✓	✓	✓	✓		✓[d]						
Input Point Validity																																✓[e]
Curve Integrity Check																						✓										

Table 11.1: Attacks versus Countermeasures

[a] except for the Montgomery Ladder
[b] except for BRIP and the regular Shamir's trick
[c] except for the known fault
[d] only if the check is performed after the Projective-to-affine conversion
[e] only if the check is performed after a randomization of the base point

Conclusion and Perspectives

This thesis is a survey on the physical attacks and countermeasures on ECC. The feasibility of each attack depending on the context is detailed. The cost of each countermeasure are detailed as well. Surveys on physical cryptanalysis usually display the attacks and countermeasures separately. We tried a different approach. The attacks and countermeasures are exhibited with a tree structure to clearly indicates if an attack has been introduced against a specific countermeasure or if it more general. Similarly, we can see if a countermeasure has been proposed against a specific attack. A synthesis in tabular form of attacks and countermeasures is given at the end. For this one, we clearly separate the attacks from the countermeasures.

We introduced new attacks called Same-Values Analysis. The attacks are named after the same principle: they all take advantage of same values occurring within an Elliptic Curve Scalar Multiplication (ECSM). They differ from the targeted implementation or from the method used to detect the occurrence of the same values.

The classical SVA consists in choosing the suitable base point so that same values occur only if some condition of the scalar is met. This attack, originally proposed as a vertical attack, was extended into a horizontal analysis. We also used the occurrence of same values to target some existing countermeasures such as the Unified Formulæ and the Side-Channel Atomicity countermeasures, with a single trace.

Depending on the attack, we used different existing statistical methods to detect the occurrence of same values. For the Horizontal SVA on the Atomicity Countermeasure, we also used a new method inspired from the Big Mac attack. For each method, we gave experimental results to validate the attacks.

With this new kind of attacks, we showed that the occurrence of same values within an ECSM can be exploited by the attacker. This concept can probably be modified and refined to target other implementations, and even to other asymmetric cryptosystems. Against these attacks, we are currently studying on new methods to ensure that the occurrence of same values is not possible.

We also introduced a new Differential Fault Attack. As opposed to previous fault attacks, where the fault is induced at the beginning or during the ECSM, our attack targets the final conversion process from projective to affine coordinates. Such faults permit to recover some information of the missing Z coordinate of the projective or Jacobian coordinates systems. This makes it possible to retrieve a few bits of the scalar from a method presented by Naccache et al. at Eurocrypt 2005.

Also, a new countermeasure against the Refined Side-Channel Analysis (RSCA) is presented. The RSCA relies on the occurrence of a particular point, namely the points of the form $(0, y)$. We proposed to use an isomorphism between elliptic curves to control the inconvenient point.

Its occurrence does not reveal anything about the scalar. Because of the isomorphism, elliptic curve formulæ are updated. Under certain assumptions that we clearly detailed, the new formulæ are in fact more efficient than the regular ones.

Finally, we showed that some differential fault attacks are feasible on the ECDSA. These attacks need the comparison of one erroneous result and the corresponding correct result. In ECDSA, the scalar changes at each new signature. Intuitively, one can think that the signature scheme is naturally immune against DFAs. We showed that this is not the case for some attacks because of some properties of the ECDSA.

This thesis gave a state-of-the-art on attacks and countermeasures on ECC. A detailed description of the attacks and the cost of the countermeasure is useful for the designer trying to protect his implementation. In the introduction, we emphasized that the topic on side-channel and fault analysis is a cat-and-mouse game. In the future, new attacks will necessarily emerge, targeting a specific implementation or a countermeasure, or it will be more general. New countermeasures will appear as well. The structure tree proposed and the synthesis at the end are suitable to easily incorporate new attacks and countermeasures.

It would be interesting to extend this work to other asymmetric cryptosystems such as RSA or pairing-based cryptography. Another method would be to give a single state-of-the-art for the different asymmetric cryptosystems in some way. Indeed there are strong similarities between the implementations. First, the same arithmetic module is generally used for RSA, ECC and pairing-based cryptography. Secondly, Elliptic Curve Scalar Multiplication methods for both ECC and pairing-based cryptography are similar to the modular exponentiation methods for RSA. Some attacks and countermeasures would also be similar.

Publications

[DGH+12] J.-L. Danger, S. Guilley, P. Hoogvorst, C. Murdica and D. Naccache, *Low-Cost Countermeasure against RPA*. Proceedings of CARDIS'12, LNCS vol. 7771, Springer, 2013, pp. 106-122.

[DGH+13] J.-L. Danger, S. Guilley, P. Hoogvorst, C. Murdica and D. Naccache, *A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards*. *J. Cryptographic Engineering* vol. 3, iss. 4, 2013, pp 241-265.

[MMNT13] D. Maimut, C. Murdica, D. Naccache and M. Tibouchi *Fault Attacks on Projective-to-Affine Coordinates Conversion*. Proceedings of COSADE'13, LNCS vol. 7864, Springer, 2013, pp. 46-61.

[MGD+12] C. Murdica, S. Guilley, J.-L. Danger, P. Hoogvorst and D. Naccache, *Same Values Power Analysis Using Special Points on Elliptic Curves*. Proceedings of COSADE'12, LNCS vol. 7275, Springer, 2012, pp. 183-198.

References

[AT03] T. Akishita and T. Takagi, *Zero-Value Point Attacks on Elliptic Curve Cryptosystem*. Proceedings of ISC'03, LNCS vol. 2851, Springer, 2003, pp. 218-233.

[ANSI X9.62] ANSI X9.62, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. ANSI, USA, 1998.

[ANSI X9.63] ANSI X9.63, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*. ANSI, USA, 2001.

[AVFM07] F. Amiel, K. Villegas, B. Feix and L. Marcel, *Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis*. FDTC'07, IEEE Computer Society, 2007, pp. 92-102.

[Baj98] J.-C. Bajard, *An RNS Montgomery Modular Multiplication Algorithm*. *IEEE Trans. Computers* vol. 47, iss. 7, 1998, pp. 766-776.

[BCN+06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, *The Sorcerer's Apprentice Guide to Fault Attacks*. Proceedings of the IEEE'06, vol. 94, iss. 2, pp. 370-382.

[Bar86] P. Barrett, *Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor*. Proceedings of CRYPTO'86, LNCS vol. 263, Springer, 1987, pp. 311-323.

[BJPW13a] A. Bauer, É. Jaulmes, E. Prouff and J. Wild, *Horizontal and Vertical Side-Channel Attacks against Secure* RSA *Implementations*. Proceedings of CT-RSA'13, LNCS vol. 7779, Springer, 2013, pp. 1-17.

[BJPW13b] A. Bauer, É. Jaulmes, E. Prouff and J. Wild, *Horizontal Collision Correlation Attack on Elliptic Curves*. To appear in the proceedings of SAC'13, LNCS.

[BL04] D. J. Bernstein and T. Lange, *Explicit-formulas database*. 2004, `hyperelliptic.org/EFD`.

[BMM00] I. Biehl, B. Meyer and V. Müller, *Differential Fault Attacks on Elliptic Curve Cryptosystems*. Proceedings of CRYPTO'00, LNCS vol. 1880, Springer, 2000, pp. 131-146.

[BS97] E. Biham and A. Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*. Proceedings of CRYPTO'97, LNCS vol. 1294, Springer-Verlag, 1997, pp. 513-525.

[BSS99] I. Blake, G. Seroussi and N. Smart, *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[BOS06] J. Blömer, M. Otto and J.-P. Seifert, *Sign Change Fault Attacks on Elliptic Curve Cryptosystems*. Proceedings of FDTC'06, LNCS vol. 4236, Springer, 2006, pp. 36-52.

[BDL97] D. Boneh, R. DeMillo and R. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)*. Proceedings of EUROCRYPT'97, LNCS vol. 1233, Springer-Verlag, 1997, pp. 37-51.

[BHT09] A. Boscher, H. Handschuh and E. Trichina, *Blinded Fault Resistant Exponentiation Revisited*. FDTC'09, IEEE Computer Society, 2009, pp. 3-9.

[BJ02] É. Brier and M. Joye, *Weierstraß Elliptic Curves and Side-Channel Attacks*. Proceedings of PKC'02, LNCS vol. 2274, Springer, 2002, pp. 335-345.

[CJ03] M. Ciet and M. Joye, *(Virtually) Free Randomization Techniques for Elliptic Curve Cryptography*. Proceedings of ICIS'03, LNCS vol. 2836, Springer, 2003, pp. 348-359.

[CJ05] M. Ciet and M. Joye, *Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults*. *Des. Codes Cryptography* vol. 36, iss. 1, 2005, pp. 33-43.

[CRR02] S. Chari, J. R. Rao and P. Rohatgi, *Template Attacks*. Proceedings of CHES'02, LNCS vol. 2523, Springer, 2003, pp. 13-28.

[CCJ04] B. Chevallier-Mames, M. Ciet and M. Joye, *Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity*. *IEEE Trans. Computers* vol. 53, iss. 6, 2004, pp. 460-468.

[CC86] D. V. Chudnovsky and G. V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*. *Adv. Appl. Math.* vol. 7, iss. 4, 1986, pp. 385-502.

[CFG+12] C. Clavier, B. Feix, G. Gagnerot, C. Giraud, M. Rousselet and V. Verneuil, *ROSETTA for Single Trace Analysis*. Proceedings of INDOCRYPT'12, LNCS vol. 7668, Springer, 2012, pp. 140-155.

[CFG+11] C. Clavier, B. Feix, G. Gagnerot, M. Rousselet and V. Verneuil, *Improved Collision-Correlation Power Analysis on First Order Protected* AES. Proceedings of CHES'11, LNCS vol. 6917, Springer, 2011, pp. 49-62.

[CFG+10] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet and V. Verneuil, *Horizontal Correlation Analysis on Exponentiation*. Proceedings of ICICS'10, LNCS vol. 6476, Springer, 2010, pp. 46-61.

[CJ01] C. Clavier and M. Joye, *Universal Exponentiation Algorithm*. Proceedings of CHES'01, LNCS vol. 2162, Springer, 2001, pp. 300-308.

[CFA+06] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2006.

[CMO98] H. Cohen, A. Miyaji and T. Ono, *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*. Proceedings of ASIACRYPT'98, LNCS vol. 1514, Springer, 1998, pp. 51-65.

[Cor99] J.-S. Coron, *Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems*. Proceedings of CHES'99, LNCS vol. 1717, Springer, 1999, pp. 292-302.

[DKL+98] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater and J.-L. Willems, *A Practical Implementation of the Timing Attack*. Proceedings of CARDIS'98, LNCS vol. 1820, Springer, 2000, pp. 167-182.

[DH11] A. Dominguez-Oviedo and M. A. Hansan, *Algorithm-level error detection for Montgomery ladder-based* ECSM. *J. Cryptographic Engineering* vol. 1, iss. 1, 2011, pp 57-69.

[FGV11] J. Fan, B. Gierliches and F. Vercauteren, *To Infinity and Beyond: Combined Attack on* ECC *Using Points of Low Order*. Proceedings of CHES'11, LNCS vol. 6917, Springer, 2011, pp. 143-159.

[FGD+10] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel and I. Verbauwhede, *State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures*. Proceedings of HOST'10, IEEE Computer Society, 2010, pp. 76-87.

[FPPR12] J.-C. Faugères, L. Perret, C. Petit and G. Renault, *Improving the Complexity of Index Calculus Algorithms in Elliptic Curves over Binary Fields*. Proceedings of EUROCRYPT'12, LNCS vol. 7237, Springer, 2012, pp. 27-44.

[FIPS186-3] FIPS PUB 186-3. *Digital Signature Standard (*DSS*)*. NIST, USA, 2009.

[FLRV08] P.-A. Fouque, R. Lercier, D. Réal and F. Valette, *Fault Attack on Elliptic Curve Montgomery Ladder Implementation*. FDTC'08, IEEE Computer Society, 2008, pp. 92-98.

[FRVD08] P.-A. Fouque, D. Réal, F. Valette and M. Drissi, *The Carry Leakage on the Randomized Exponent Countermeasure*. Proceedings of CHES'08, LNCS vol. 5154, Springer, 2008, pp. 198-213.

[FSW02] P.-A. Fouque, J. Stern and J.G. Wackers, *CryptoComputing with Rationals*. Proceedings of Financial Cryptography'02, LNCS vol. 2357, Springer, 2003, pp. 136-146.

[FV03] P.-A. Fouque and F. Valette, *The Doubling Attack - Why Upwards Is Better than Downwards*. Proceedings of CHES'03, LNCS vol. 2779, Springer, 2003, pp. 269-280.

[Gir06] C. Giraud, *An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis*. *IEEE Trans. Computers* vol. 55, iss. 9, 2006, pp. 1116-1120.

[GV10] C. Giraud and V. Verneuil, *Atomicity Improvement for Elliptic Curve Scalar Multiplication*. Proceedings of CARDIS'10, LNCS vol. 6035, Springer, 2010, pp. 80-101.

[Gou03] L. Goubin, *A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems*. Proceedings of PKC'03, LNCS vol. 2567, Springer-Verlag, 2002, pp. 199-210.

[GJM10] R. R. Goundar, M. Joye and A. Miyaji, *Co-Z Addition Formulæ and Binary Ladders on Elliptic Curves - (Extended Abstract)*. Proceedings of CHES'10, LNCS vol. 6225, Springer-Verlag, 2010, pp. 65-79.

[GJM+11] R. R. Goundar, M. Joye, A. Miyaji, M. Rivain and A. Venelli, *Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic*. *J. Cryptographic Engineering* vol. 1, iss. 2, 2011, pp 161-176.

[Has36] H. Hasse, *Zur Theorie der abstrakten elliptischen Funktionenkörper III*. *J. Reine Angew. Math.* vol. 1936, iss. 175, pp 193–208.

[HQ00] G. Hachez and J.-J. Quisquater, *Montgomery Exponentiation with no Final Subtractions: Improved Results*. Proceedings of CHES'00, LNCS vol. 1965, Springer, 2000, pp. 293-301

[HMV03] D. Hankerson, A. J. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., 2003.

[HS01] N. Howgrave-Graham and N. Smart, *Lattice Attacks on Digital Signature Schemes*. *Des. Codes Cryptography* vol. 23, iss. 3, 2001, pp. 283-290.

[HJS11] M. Hutter, M. Joye, and Y. Sierra, *Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation*. Proceedings of AFRICACRYPT'11, LNCS vol. 6737, Springer, 2011, pp. 170-187.

[IIT02] K. Itoh, T. Izu and M. Takenaka, *Address-Bit Differential Power Analysis of Cryptographic Schemes* OK-ECDH *and* OK-ECDSA. Proceedings of CHES'02, LNCS vol. 2523, Springer, 2003, pp. 129-143.

[IIT03] K. Itoh, T. Izu and M. Takenaka, *A Practical Countermeasure against Address-Bit Differential Power Analysis*. Proceedings of CHES'03, LNCS vol. 2779, Springer, 2003, pp. 382-396.

[IIT04] K. Itoh, T. Izu and M. Takenaka, *Efficient Countermeasures against Power Analysis for Elliptic Curve Cryptosystems*. CARDIS'04, Kluwer, 2004, pp. 99-114.

[IMT02] T. Izu, B. Möller and T. Takagi, *Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks*. INDOCRYPT'02, LNCS vol. 2551, Springer, 2002, pp. 296-313.

[IT03] T. Izu and T. Takagi, *Exceptional Procedure Attack on Elliptic Curve Cryptosystems*. Proceedings of PKC'03, LNCS vol. 2567, Springer, 2003, pp. 224-239.

[IISO10] M. Izumi, J. Ikegami, K. Sakiyama and K. Ohta, *Improved countermeasure against Address-bit DPA for ECC scalar multiplication*. DATE'10, IEEE, 2010, pp. 981-984.

[Joy07] M. Joye, *Highly Regular Right-to-Left Algorithms for Scalar Multiplication*. Proceedings of CHES'07, LNCS vol. 4727, Springer, 2007, pp. 135-147.

[JT01] M. Joye and C. Tymen, *Protections against Differential Analysis for Elliptic Curve Cryptography*. Proceedings of CHES'01, LNCS vol. 2162, Springer, 2001, pp. 377-390.

[JY02] M. Joye and S.-M. Yen, *The Montgomery Powering Ladder*. Proceedings of CHES'02, LNCS vol. 2162, Springer, 2003, pp. 291-302.

[Kob87] N. Koblitz, *Elliptic Curve Cryptosystems*. *Mathematics of Computation* vol. 48, iss. 177, 1987, pp 203-209.

[KA96] C. K. Koç and T. Acar, *Analyzing and Comparing Montgomery Multiplication Algorithms*. *Micro, IEEE* vol. 16, iss. 3, 1996, pp. 26-33.

[Koc96] P. C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman,* RSA, DSS, *and Other Systems*. Proceedings of CRYPTO'96, LNCS vol. 1109, Springer, 1996, pp. 104-113.

[LLL82] A.K. Lenstra, H.W. Lenstra and L. Lovàsz, *Factoring polynomials with rational coefficients*. *Math. Ann.* vol. 261, 1982, pp. 515-534.

[MMM04] H. Mamiya, A. Miyaji and H. Morimoto, *Efficient Countermeasures against RPA, DPA, and SPA*. Proceedings of CHES'04, LNCS vol. 3156, Springer, 2004, pp. 343-356.

[MO08] M. Medwed and E. Oswald, *Template Attacks on* ECDSA. Proceedings of WISA'08, LNCS vol. 5379, Springer, 2009, pp. 14-27.

[Mel07] N. Meloni, *New Point Addition Formulae for* ECC *Applications*. Proceedings of WAIFI'07, LNCS vol. 4547, Springer, 2007, pp. 189-201.

[Mil85] V. S. Miller, *Use of elliptic curves in cryptography*. Proceedings of CRYPTO'85, LNCS vol. 218, Springer, 1985, pp. 417-426.

[Mon85] P. L. Montgomery, *Modular Multiplication without Trial Division*. *Mathematics of Computation* vol. 44, iss. 170, 1985, pp 519-521.

[MV06] F. Muller and F. Valette, *High-Order Attacks Against the Exponent Splitting Protection*. Proceedings of PKC'06, LNCS vol. 3958, Springer, 2006, pp. 315-329.

[NSS04] D. Naccache, N. Smart and J. Stern, *Projective Coordinates Leak*. Proceedings of EUROCRYPT'04, LNCS vol. 3027, Springer, 2004, pp. 257-267.

[PW04] V.Y. Pan and X. Wang, *On Rational Number Reconstruction and Approximation*. *SIAM J. Comput.* vol. 33, iss. 2, 2004, pp. 502-503.

[Pol78] J. Pollard, *Monte Carlo methods for Index Computation (mod p)*. *Mathematics of Computation* vol. 32, iss. 143, 1978, pp 918-924.

[Qui90] J.J. Quisquater, *Procédé de codage selon la méthode dite* RSA *par un microcontrôleur et dispositifs utilisant ce procédé*. Demande de brevet français, No de dépôt 90 02274, 1990.

[Qui91] J.J. Quisquater, *Encoding system according to the so-called* RSA *method, by means of a microcontroller and arrangement implementing this system*. U.S. Patent #5,166,978, 1991.

[SASEBO] *Side-channel Attack Standard Evaluation Board (SASEBO)*. `http://www.rcis.aist.go.jp/special/SASEBO/`

[SST04] H. Sato, D. Schepers and T. Takagi, *Exact Analysis of Montgomery Multiplication*. Proceedings of INDOCRYPT'04, LNCS vol. 3348, Springer, 2004, pp. 290-304.

[SWP03] K. Schramm, T.J. Wollinger and C. Paar, *A New Class of Collision Attacks and Its Application to* DES. FSE'03, LNCS vol. 2887, Springer-Verlag, 2003, pp. 206-222.

[Sha71] D. Shanks, *Class Number, a Theory of Factorization and Genera*. *Proceedings of Symposia in Pure Mathematics* vol. 20, 1971, pp. 415-440.

[Str64] E. G. Straus, *Addition chains of vectors (problem 5125)*. THE AMERICAN MATHEMATICAL MONTHLY vol. 71, num. 7, 1964, pp. 806-808.

[ST06] D. Stebila and N. Thériault, *Unified Point Addition Formulæ and Side-Channel Attacks*. Proceedings of CHES'06, LNCS vol. 4249, Springer, 2006, pp. 354-368.

[SAGE12] W. A. Stein *et al. Sage Mathematics Software (Version 5.0)*. The Sage Development Team, 2012, `http://www.sagemath.org`.

[TE02] E. Trichina and A. Bellezza, *Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks*. Proceedings of CHES'02, LNCS vol. 2523, Springer, 2002, pp. 98-113.

[Val91] B. Vallée, *Gauss' Algorithm Revisited*. *Journal of Algorithms* vol. 12, iss. 4, 1991, pp. 556-572.

[Ver12] V. Verneuil, *Cryptographie à base de courbes elliptiques et sécurité de composants embarqués*. Ph.D. thesis, Université de Bordeaux, 2012.

[Wal99] C. D. Walter, *Montgomery's Multiplication Technique: How to Make It Smaller and Faster*. Proceedings of CHES'99, LNCS vol. 1717, Springer, 1999, pp. 80-93.

[Wal01] C. D. Walter, *Sliding Windows Succumbs to Big Mac Attack*. Proceedings of CHES'01, LNCS vol. 2162, Springer, 2001, pp. 286-299.

[Wal04] C. D. Walter, *Simple Power Analysis of Unified Code for* ECC *Double and Add*. Proceedings of CHES'04, LNCS vol. 3156, Springer, 2004, pp. 191-204.

[WP03] X. Wang and V.Y. Pan, *Acceleration of Euclidean algorithm and rational number reconstruction*. *SIAM J. Comput.* vol. 32, iss. 2, 2003, pp. 548-556.

[YJ00] S.-M. Yen and M. Joye, *Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis*. *IEEE Trans. Computers* vol. 49, iss. 9, 2000, pp. 967-970.

[YKLM01] S.-M. Yen, S. Kim, S. Lim and S.-M. Moon, *A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack*. Proceedings of ICISC'01, LNCS vol. 2288, Springer, 2001, pp. 141-427.

Elliptic Curve Formulæ with register allocation

The following appendix resumes some algorithms of Section 3.1 and, in addition, it gives the detailed registers allocation.

Classical Formulæ

Algorithm 37 ECADD (register allocation)

Input: $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$
Output: $P + Q$

$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_3 \leftarrow Z_1$
$T_4 \leftarrow X_2;\ T_5 \leftarrow Y_2;\ T_6 \leftarrow Z_2$

1:	$T_7 \leftarrow T_3 \times T_6$	$\{Z_1Z_2\}$	12:	$T_3 \leftarrow T_7 \times T_9$	$\{Z_3 = Z_1Z_2E\}$
2:	$T_8 \leftarrow T_3^2$	$\{Z_1^2\}$	13:	$T_7 \leftarrow T_9^2$	$\{E^2\}$
3:	$T_3 \leftarrow T_3 \times T_8$	$\{Z_1^3\}$	14:	$T_9 \leftarrow T_7 \times T_9$	$\{E^3\}$
4:	$T_3 \leftarrow T_5 \times T_3$	$\{D = Y_2Z_1^3\}$	15:	$T_7 \leftarrow T_1 \times T_7$	$\{AE^2\}$
5:	$T_9 \leftarrow T_4 \times T_8$	$\{B = X_2Z_1^2\}$	16:	$T_2 \leftarrow T_2 \times T_9$	$\{CE^3\}$
6:	$T_8 \leftarrow T_6^2$	$\{Z_2^2\}$	17:	$T_1 \leftarrow T_8^2$	$\{F^2\}$
7:	$T_1 \leftarrow T_1 \times T_8$	$\{A = X_1Z_2^2\}$	18:	$T_1 \leftarrow T_1 - T_9$	$\{F^2 - E^3\}$
8:	$T_8 \leftarrow T_8 \times T_6$	$\{Z_2^3\}$	19:	$T_1 \leftarrow T_1 - T_7$	$\{F^2 - E^3 - AE^2\}$
9:	$T_2 \leftarrow T_2 \times T_8$	$\{C = Y_1Z_2^3\}$	20:	$T_1 \leftarrow T_1 - T_7$	$\{X_3\}$
10:	$T_9 \leftarrow T_9 - T_1$	$\{E = B - A\}$	21:	$T_7 \leftarrow T_7 - T_1$	$\{AE^2 - X_3\}$
11:	$T_8 \leftarrow T_3 - T_2$	$\{F = D - C\}$	22:	$T_8 \leftarrow T_8 \times T_7$	$\{F(AE^2 - X_3)\}$
			23:	$T_2 \leftarrow T_8 - T_2$	$\{Y_3\}$

return (T_1, T_2, T_3)

Remark .1. T_4, T_5, T_6 are not modified. The operation $P \leftarrow \text{ECADD}(P, Q)$ can be done with 3 extra temporary registers without modifying the coordinates of Q.

Algorithm 38 mECADD (register allocation)

Input: $P = (X_1, Y_1, Z_1), Q = (x_2, y_2)$
Output: $P + Q$
$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_3 \leftarrow Z_1$
$T_4 \leftarrow x_2;\ T_5 \leftarrow y_2$

1:	$T_7 \leftarrow T_3^2$	$\{Z_1^2\}$	10:	$T_7 \leftarrow T_7 \times T_1$	$\{X_1E^2\}$
2:	$T_9 \leftarrow T_4 \times T_7$	$\{B = x_2Z_1^2\}$	11:	$T_2 \leftarrow T_2 \times T_9$	$\{Y_1E^3\}$
3:	$T_7 \leftarrow T_7 \times T_3$	$\{Z_1^3\}$	12:	$T_1 \leftarrow T_8^2$	$\{F^2\}$
4:	$T_7 \leftarrow T_7 \times T_5$	$\{D = y_2Z_1^3\}$	13:	$T_1 \leftarrow T_1 - T_9$	$\{F^2 - E^3\}$
5:	$T_9 \leftarrow T_9 - T_1$	$\{E = B - X_1\}$	14:	$T_1 \leftarrow T_1 - T_7$	$\{F^2 - E^3 - X_1E^2\}$
6:	$T_8 \leftarrow T_7 - T_2$	$\{F = D - Y_1\}$	15:	$T_1 \leftarrow T_1 - T_7$	$\{X_3\}$
7:	$T_3 \leftarrow T_3 \times T_9$	$\{Z_3 = Z_1E\}$	16:	$T_7 \leftarrow T_7 - T_1$	$\{X_1E^2 - X_3\}$
8:	$T_7 \leftarrow T_9^2$	$\{E^2\}$	17:	$T_8 \leftarrow T_8 \times T_7$	$\{F(X_1E^2 - X_3)\}$
9:	$T_9 \leftarrow T_9 \times T_7$	$\{E^3\}$	18:	$T_2 \leftarrow T_8 - T_2$	$\{Y_3\}$

return (T_1, T_2, T_3)

Algorithm 39 ECDBL (register allocation)

Input: $P = (X_1, Y_1, Z_1)$, elliptic curve parameter a
Output: $2P$
$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_3 \leftarrow Z_1$

1:	$T_6 \leftarrow T_2^2$	$\{Y_1^2\}$	12:	$T_5 \leftarrow T_5 \times a$	$\{aZ_1^4\}$
2:	$T_6 \leftarrow T_6 + T_6$	$\{A = 2Y_1^2\}$	13:	$T_5 \leftarrow T_5 + T_1$	$\{C = 3X_1^2 + aZ_1^4\}$
3:	$T_4 \leftarrow T_1 \times T_6$	$\{AX_1\}$	14:	$T_1 \leftarrow T_5^2$	$\{C^2\}$
4:	$T_4 \leftarrow T_4 + T_4$	$\{B = 2AX_1\}$	15:	$T_1 \leftarrow T_1 - T_4$	$\{C^2 - B\}$
5:	$T_1 \leftarrow T_1^2$	$\{X_1^2\}$	16:	$T_1 \leftarrow T_1 - T_4$	$\{C^2 - 2B = X_3\}$
6:	$T_5 \leftarrow T_1 + T_1$	$\{2X_1^2\}$	17:	$T_6 \leftarrow T_6^2$	$\{A^2\}$
7:	$T_1 \leftarrow T_1 + T_5$	$\{3X_1^2\}$	18:	$T_6 \leftarrow T_6 + T_6$	$\{D = 2A^2\}$
8:	$T_5 \leftarrow T_3^2$	$\{Z_1^2\}$	19:	$T_2 \leftarrow T_4 - T_1$	$\{B - X_3\}$
9:	$T_3 \leftarrow T_3 \times T_2$	$\{Y_1Z_1\}$	20:	$T_2 \leftarrow T_2 \times T_5$	$\{C(B - X_3)\}$
10:	$T_3 \leftarrow T_3 + T_3$	$\{Z_3 = 2Y_1Z_1\}$	21:	$T_2 \leftarrow T_2 - T_6$	$\{Y_3\}$
11:	$T_5 \leftarrow T_5^2$	$\{Z_1^4\}$			

return (T_1, T_2, T_3)

Co-Z Formulæ

Algorithm 40 ZADDU (register allocation)

Input: $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z)$
Output: (R, S) with $R = P + Q$ and $S = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z)$ with $\lambda = X_1 - X_2$
$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_3 \leftarrow X_2;\ T_4 \leftarrow Y_2;\ T_5 \leftarrow Z$

1:	$T_6 \leftarrow T_1 - T_3$	$\{X_1 - X_2\}$	8:	$T_4 \leftarrow T_2 \times T_4$	$\{A_1 = Y_1(W_1 - W_2)\}$
2:	$T_5 \leftarrow T_5 \times T_6$	$\{Z(X_1 - X_2)\}$	9:	$T_1 \leftarrow T_6^2$	$\{D = (Y_1 - Y_2)^2\}$
3:	$T_6 \leftarrow T_6^2$	$\{C = (X_1 - X_2)^2\}$	10:	$T_1 \leftarrow T_1 - T_3$	$\{D - W_1\}$
4:	$T_7 \leftarrow T_3 \times T_6$	$\{W_2 = X_2 C\}$	11:	$T_1 \leftarrow T_1 - T_7$	$\{X_3 = D - W_1 - W_2\}$
5:	$T_3 \leftarrow T_1 \times T_6$	$\{W_1 = X_1 C\}$	12:	$T_2 \leftarrow T_3 - T_1$	$\{W_1 - X_3\}$
6:	$T_6 \leftarrow T_2 - T_4$	$\{Y_1 - Y_2\}$	13:	$T_2 \leftarrow T_2 \times T_6$	$\{Y_3 + A_1\}$
7:	$T_4 \leftarrow T_3 - T_7$	$\{W_1 - W_2\}$	14:	$T_2 \leftarrow T_2 - T_4$	$\{Y_3\}$

return $((T_1, T_2, T_5), (T_3, T_4, T_5))$

Algorithm 41 ZADDC (register allocation)

Input: $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z)$
Output: (R, S) with $R = P + Q$, $S = P - Q$
$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_3 \leftarrow X_2;\ T_4 \leftarrow Y_2;\ T_5 \leftarrow Z$

1:	$T_6 \leftarrow T_1 - T_3$	$\{X_1 - X_2\}$	11:	$T_1 \leftarrow T_3^2$	$\{D_1 = (Y_1 - Y_2)^2\}$
2:	$T_5 \leftarrow T_5 \times T_6$	$\{Z(X_1 - X_2)\}$	12:	$T_1 \leftarrow T_1 - T_6$	$\{X_3 = D_1 - W_1 - W_2\}$
3:	$T_6 \leftarrow T_6^2$	$\{C = (X_1 - X_2)^2\}$	13:	$T_2 \leftarrow T_8 - T_1$	$\{W_1 - X_3\}$
4:	$T_8 \leftarrow T_1 \times T_6$	$\{W_1 = X_1 C\}$	14:	$T_2 \leftarrow T_2 \times T_3$	$\{Y_3 + A_1\}$
5:	$T_6 \leftarrow T_3 \times T_6$	$\{W_2 = X_2 C\}$	15:	$T_2 \leftarrow T_2 - T_7$	$\{Y_3\}$
6:	$T_7 \leftarrow T_8 - T_6$	$\{W_1 - W_2\}$	16:	$T_3 \leftarrow T_4^2$	$\{D_2 = (Y_1 + Y_2)^2\}$
7:	$T_6 \leftarrow T_8 + T_6$	$\{W_1 + W_2\}$	17:	$T_3 \leftarrow T_3 - T_6$	$\{X_4 = D_2 - W_1 - W_2\}$
8:	$T_7 \leftarrow T_7 \times T_2$	$\{A_1 = Y_1(W_1 - W_2)\}$	18:	$T_6 \leftarrow T_8 - T_3$	$\{W_1 - X_4\}$
9:	$T_3 \leftarrow T_2 - T_4$	$\{Y_1 - Y_2\}$	19:	$T_4 \leftarrow T_4 \times T_6$	$\{Y_4 + A_1\}$
10:	$T_4 \leftarrow T_2 + T_4$	$\{Y_1 + Y_2\}$	20:	$T_4 \leftarrow T_4 - T_7$	$\{Y_4\}$

return $((T_1, T_2, T_5), (T_3, T_4, T_5))$

Algorithm 42 ZADDU-ISO-SHIFTING (register allocation)

Input: $P' = (X_1, Y_1, Z, U = a_2' Z^2), Q' = (0, Y_2, Z, U)$
Output: (R', S') with $R' = P' + Q'$ and $S' = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z, a_2'(\lambda Z)^2)$ with $\lambda = X_1$
$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_4 \leftarrow Y_2;\ T_5 \leftarrow Z;\ T_6 \leftarrow U$

1:	$T_5 \leftarrow T_5 \times T_1$	$\{Z_3 = ZX_1\}$	7:	$T_1 \leftarrow T_7^2$	$\{D = (Y_1 - Y_2)^2\}$
2:	$T_7 \leftarrow T_1^2$	$\{C = X_1^2\}$	8:	$T_1 \leftarrow T_1 - T_6$	$\{D - U_3\}$
3:	$T_3 \leftarrow T_1 \times T_7$	$\{W_1 = X_1 C\}$	9:	$T_1 \leftarrow T_1 - T_3$	$\{X_3 = D - W_1 - U_3\}$
4:	$T_6 \leftarrow T_6 \times T_7$	$\{U_3 = UC\}$	10:	$T_2 \leftarrow T_3 - T_1$	$\{W_1 - X_3\}$
5:	$T_7 \leftarrow T_2 - T_4$	$\{Y_1 - Y_2\}$	11:	$T_2 \leftarrow T_2 \times T_7$	$\{Y_3 + A_1\}$
6:	$T_4 \leftarrow T_2 \times T_3$	$\{A_1 = Y_1 W_1\}$	12:	$T_2 \leftarrow T_2 - T_4$	$\{Y_3\}$

return $((T_1, T_2, T_5, T_6), (T_3, T_4, T_5, T_6))$

Algorithm 43 ZADDC-ISO-SHIFTING (register allocation)

Input: $P' = (X_1, Y_1, Z, U = a_2' Z^2), Q' = (X_2, Y_2, Z, U)$ such that $x_{P'-Q'} = 0$.
Output: (R', S) with $R' = P' + Q'$, $S' = P' - Q'$
$T_1 \leftarrow X_1;\ T_2 \leftarrow Y_1;\ T_3 \leftarrow X_2;\ T_4 \leftarrow Y_2;\ T_5 \leftarrow Z;\ T_6 \leftarrow U$

1:	$T_7 \leftarrow T_1 - T_3$	$\{X_1 - X_2\}$	11:	$T_4 \leftarrow T_2 + T_4$	$\{Y_1 + Y_2\}$
2:	$T_5 \leftarrow T_5 \times T_7$	$\{Z(X_1 - X_2)\}$	12:	$T_4 \leftarrow T_4 \times T_3$	$\{(Y_1 + Y_2)W_1\}$
3:	$T_7 \leftarrow T_7^2$	$\{C = (X_1 - X_2)^2\}$	13:	$T_2 \leftarrow T_7^2$	$\{D = (Y_1 - Y_2)^2\}$
4:	$T_6 \leftarrow T_6 \times T_7$	$\{U_3 = UC\}$	14:	$T_2 \leftarrow T_2 - T_6$	$\{D - U_3\}$
5:	$T_8 \leftarrow T_3 \times T_7$	$\{W_2 = X_2 C\}$	15:	$T_1 \leftarrow T_2 - T_1$	$\{X_3 = D - W_1 - W_2 - U_3\}$
6:	$T_3 \leftarrow T_1 \times T_7$	$\{W_1 = X_1 C\}$	16:	$T_3 \leftarrow T_3 - T_1$	$\{W_1 - X_3\}$
7:	$T_1 \leftarrow T_3 + T_8$	$\{W_1 + W_2\}$	17:	$T_2 \leftarrow T_7 \times T_3$	$\{Y_3 + A_1\}$
8:	$T_8 \leftarrow T_3 - T_8$	$\{W_1 - W_2\}$	18:	$T_2 \leftarrow T_2 - T_8$	$\{Y_3\}$
9:	$T_8 \leftarrow T_8 \times T_2$	$\{A_1 = Y_1(W_1 - W_2)\}$	19:	$T_4 \leftarrow T_4 - T_8$	$\{Y_4\}$
10:	$T_7 \leftarrow T_2 - T_4$	$\{Y_1 - Y_2\}$	20:	$T_3 \leftarrow 0$	

return $((T_1, T_2, T_5, T_6), (T_3, T_4, T_5, T_6))$