Efficacité et sécurité contre les attaques par canaux auxiliaires des implémentations de RSA utilisant un système modulaire de représentation (RNS)

Fabrice Ben Hamouda
Sous la direction d'Elisabeth Oswald et de Dan Page

ENS – Bristol Cryptography Group

21 Septembre 2011

Introduction — Motivations

Les cartes à puces sont omniprésentes :
carte bancaire, carte vitale, passe Navigo, carte d'accès à l'ENS, ...

Les cartes à puce stockent généralement une clé secrète et utilisent des algorithmes cryptographiques pour permettre leur authentification (et la confidentialité des transferts de données).

Introduction — Motivations

Les cartes à puces sont omniprésentes :
carte bancaire, carte vitale, passe Navigo, carte d'accès à l'ENS, ...

Les cartes à puce stockent généralement une clé secrète et utilisent des algorithmes cryptographiques pour permettre leur authentification (et la confidentialité des transferts de données).

Parmi ces algorithmes cryptographiques, RSA est très utilisé. Il consiste grossièrement en une exponentiation modulaire de grands nombres :

$$y = x^d \mod N$$

- d : secret
- x, N : connus

Introduction — Motivations

Il y a deux défis principaux lors de l'implémentation d'algorithmes cryptographiques dans les cartes à puce :

- efficacité (temps, mémoire) de ces algorithmes cryptographiques
- sécurité :
  - théorique de l'algorithme
  - de l'implémentation elle-même
    $\longrightarrow$ contre les attaques par canaux auxiliaires

Introduction — Attaques par canaux auxiliaires

Les attaques par canaux auxiliaires consistent récupérer la clé secrète en utilisant des fuites d'information (lors d'un calcul cryptographique) liées à l'implémentation utilisée :

- temps de calcul (Paul Kocher, 1996)
- consommation électrique (Paul Kocher, 1998)
- rayonnement électromagnétique
- comportement de la carte lors de l'introduction de fautes

Introduction — Attaques par canaux auxiliaires

Les attaques par canaux auxiliaires consistent récupérer la clé secrète en utilisant des fuites d'information (lors d'un calcul cryptographique) liées à l'implémentation utilisée :

- temps de calcul (Paul Kocher, 1996)
- consommation électrique (Paul Kocher, 1998) : SPA et DPA
- rayonnement électromagnétique
- comportement de la carte lors de l'introduction de fautes

Introduction — Objectifs

Nous nous intéresserons à l'implémentation de RSA sur des microprocesseurs 32 bits.

Les instructions assembleur de tels microprocesseurs peuvent uniquement manipuler des nombres de 32 bits. Les grands entiers doivent être représentés par plusieurs entiers machine.

Introduction — Objectifs

Nous nous intéresserons à l'implémentation de RSA sur des microprocesseurs 32 bits.

Les instructions assembleur de tels microprocesseurs peuvent uniquement manipuler des nombres de 32 bits. Les grands entiers doivent être représentés par plusieurs entiers machine.

Nous utiliserons un système modulaire de représentation (RNS).

Plan

1. Système modulaire de représentation (RNS)
   - Introduction
   - Réduction de Montgomery classique
   - Réduction de Montgomery avec RNS
2. Efficacité des implémentations proposées
   - Pseudo-nombres de Mersenne et opérations efficaces sur les canaux
   - x86_64
   - ARM
   - Nios II
3. Attaques SPA et DPA
   - Introduction
   - Randomisation intra-base
   - Coût des contre-mesures

Introduction — Exponentiation rapide et RSA

L'exponentiation modulaire de RSA peut être implémentée par un algorithme d'exponentiation rapide :

Require: x est le message, N est le module, $d = \sum_{i=0}^{k-1} d_i 2^i$ est l'exposant privé ($d_i \in \{0, 1\}$, $d_{k-1} = 1$)

Ensure: $y = x^d \mod N$

1: $y \leftarrow x$
2: **for** $i = k - 2$ downto 0 **do**
3: $\quad y \leftarrow y \times y \mod N$
4: $\quad$ **if** $d_i = 1$ **then**
5: $\quad\quad y \leftarrow x \times y \mod N$
6: $\quad$ **end if**
7: **end for**
8: **return** y

qui requiert en moyenne approximativement $1.5k$ multiplications modulaires ($k = 1024$ pour RSA 1024).

Plan

Représentation classique des grands entiers et RNS

Soit x un entier naturel (de 1024 bits).
Il peut être représenté :

- par $n = 32$ nombres de 32 bits : $x_0, \ldots, x_{n-1}$, les chiffres de x écrits en base 2^{32}

$$x = x_{n-1}2^{32(n-1)} + \cdots + x_2 2^{64} + x_1 2^{32} + x_0$$

Représentation classique des grands entiers et RNS

Soit x un entier naturel (de 1024 bits).
Il peut être représenté :

- par $n = 32$ nombres de 32 bits : $x_0, \ldots, x_{n-1}$, les chiffres de x écrits en base 2^{32}

$$x = x_{n-1}2^{32(n-1)} + \cdots + x_2 2^{64} + x_1 2^{32} + x_0$$

ou

- par $n = 33$ nombres de 32 bits : $x_1, \ldots, x_n$, appelés canaux, tels que

$$x_1 = x \mod m_1 \qquad \ldots \qquad x_n = x \mod m_n$$

où $m_1, \ldots, m_n$ sont des nombres de 32 bits deux-à-deux premiers entre eux, suffisamment grands, appelés moduli. Ils forment une base $\mathfrak{B}_1$. Cette représentation est appelée RNS.

Justification de RNS

Soit $M_1 = \prod_{i=1}^{n} m_i$.
Posons :

- $|x|_m = x \mod m$
- $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z} = [0, m-1]$

Justification de RNS

Soit $M_1 = \prod_{i=1}^{n} m_i$.
Posons :

- $|x|_m = x \mod m$
- $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z} = [0, m-1]$

Theorem (Théorème des restes chinois (CRT))

La fonction suivante est un isomorphisme d'anneaux :

$$\left(\begin{array}{ccc} \mathbb{Z}_{M_1} & \rightarrow & \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \\ x & \mapsto & (x_1, \ldots, x_n) = (|x|_{m_1}, ..., |x|_{m_n}) \end{array} \right)$$

Justification de RNS

Soit $M_1 = \prod_{i=1}^{n} m_i$.
Posons :

- $|x|_m = x \mod m$
- $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z} = [0, m-1]$

Theorem (Théorème des restes chinois (CRT))

La fonction suivante est un isomorphisme d'anneaux :

$$\left(\begin{array}{ccc} \mathbb{Z}_{M_1} & \rightarrow & \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \\ x & \mapsto & (x_1, \ldots, x_n) = (|x|_{m_1}, ..., |x|_{m_n}) \end{array} \right)$$

De plus si $M_{1,i} = \frac{M_1}{m_i}$, *on a*

$$x = \sum_{i=1}^{n} \left| x_i M_{1,i}^{-1} \right|_{m_i} M_{1,i} \mod M.$$

Opérations avec RNS

Soient $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$ deux nombres RNS.
Grâce au CRT, pour toute opération $\odot \in \{+, -, \times, /\}$, on a

$$|x \odot y|_{M_1} = \left(|x_1 \odot y_1|_{m_1}, \ldots, |x_n \odot y_n|_{m_n}\right).$$

Complexité en temps : $O(n)$.
Ces opérations peuvent être faites en parallèle, pour chaque canal.

Opérations avec RNS

Soient $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$ deux nombres RNS.
Grâce au CRT, pour toute opération $\odot \in \{+, -, \times, /\}$, on a

$$|x \odot y|_{M_1} = \left(|x_1 \odot y_1|_{m_1}, \ldots, |x_n \odot y_n|_{m_n}\right).$$

Complexité en temps : $O(n)$.
Ces opérations peuvent être faites en parallèle, pour chaque canal.

Mais la comparaison, la division entière et la réduction modulaire sont des opérations complexes.

Opérations avec RNS

Soient $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$ deux nombres RNS.
Grâce au CRT, pour toute opération $\odot \in \{+, -, \times, /\}$, on a

$$|x \odot y|_{M_1} = \left(|x_1 \odot y_1|_{m_1}, \ldots, |x_n \odot y_n|_{m_n}\right).$$

Complexité en temps : $O(n)$.
Ces opérations peuvent être faites en parallèle, pour chaque canal.

Mais la comparaison, la division entière et la réduction modulaire sont des opérations complexes.

Comment implémenter une multiplication modulaire pour RSA ?

Réduction de Montgomery classique

Soient $R > N$ un entier premier avec N et x un entier dans $[0, RN - 1]$. La réduction de Montgomery x modulo N par rapport à R est

$$y = xR^{-1} \mod N.$$

Réduction de Montgomery classique

Soient $R > N$ un entier premier avec N et x un entier dans $[0, RN-1]$. La réduction de Montgomery x modulo N par rapport à R est

$$y = xR^{-1} \mod N.$$

Elle peut être calculée très efficacement si R est une puissance de 2.

1: $q \leftarrow x \times \left|-R^{-1}\right|_N \mod R$
2: $y \leftarrow (x + qN)/R$
3: **if** $y > N$ **then**
4: $\quad y \leftarrow y - N$
5: **end if**

Idée de la preuve :

$$x + qN \equiv 0[R]$$
$$x + qN < 2RN$$

Complexité : environ n^2 multiplications.

Utilisation de la réduction de Montgomery

On ne peut pas utiliser directement la réduction de Montgomery, sinon on obtiendrait :

$$x^d R^{-d'} \mod N$$

au lieu de

$$x^d \mod N.$$

Utilisation de la réduction de Montgomery

On ne peut pas utiliser directement la réduction de Montgomery, sinon on obtiendrait :

$$x^d R^{-d'} \mod N$$

au lieu de

$$x^d \mod N.$$

On remplace donc x par $\widetilde{x} = xR \mod N$, la représentation de Montgomery de x.
Comme

$$\widetilde{x}\widetilde{y} = (xR)(yR)R^{-1} \mod N = \widetilde{xy} \mod N,$$

on obtient, à la fin de l'exponentiation :

$$\widetilde{x^d} = x^d R$$

Réduction de Montgomery avec RNS

Pour que la réduction modulo R soit facile, on prend $R = M_1$.
La division par $R = M_1$ est impossible dans la base $\mathfrak{B}_1$. Donc on utilise une deuxième base $\mathfrak{B}_2 = (m_{n+1}, \ldots, m_{2n})$.

Réduction de Montgomery avec RNS

Pour que la réduction modulo R soit facile, on prend $R = M_1$.
La division par $R = M_1$ est impossible dans la base $\mathfrak{B}_1$. Donc on utilise une deuxième base $\mathfrak{B}_2 = (m_{n+1}, \ldots, m_{2n})$.

Require: $0 \leq x < 4N^2$ un nombre RNS dans la base $\mathfrak{B}_1 \cup \mathfrak{B}_2$.
Require: $0 < 4N < M_1, M_2$ and $\gcd(N, M_1) = 1$
Ensure: $0 \leq y < 2N$ un nombre RNS (dans la base $\mathfrak{B}_1 \cup \mathfrak{B}_2$) congruent à x modulo N.

1: $q \leftarrow x \times \left|-N^{-1}\right|_{M_1}$ dans $\mathfrak{B}_1$
2: Étendre q de $\mathfrak{B}_1$ à $\mathfrak{B}_2$
3: $y \leftarrow (x + q \times N) \times \left|M_1^{-1}\right|_{M_2}$ dans $\mathfrak{B}_2$
4: Étendre y de $\mathfrak{B}_2$ à $\mathfrak{B}_1$

Extension de base

Les étapes 1 et 3 sont en $O(n)$.
Comment effectuer les deux extensions de base ?
Quelle est leur complexité ?
Trois algorithmes possibles :

- utilisation de MRS (système en base mixte), une représentation associée à RNS
  Complexité : environ $\frac{3}{2}n^2$ multiplications
  Mémoire : $\Omega(n)$ (ou $\Omega(n^2)$) nombres de 32 bits
- utilisation du CRT selon Bajard
  Complexité : environ n^2 multiplications
  Mémoire : $\Omega(n^2)$ nombres de 32 bits
- utilisation du CRT selon Kawamura
  Complexité : environ n^2 multiplications
  Mémoire : $\Omega(n^2)$ nombres de 32 bits

Extension de base

Les étapes 1 et 3 sont en $O(n)$.
Comment effectuer les deux extensions de base ?
Quelle est leur complexité ?
Trois algorithmes possibles :

- utilisation de MRS (système en base mixte), une représentation associée à RNS
  Complexité : environ $\frac{3}{2}n^2$ multiplications
  Mémoire : $\Omega(n)$ (ou $\Omega(n^2)$) nombres de 32 bits
- utilisation du CRT selon Bajard
  Complexité : environ n^2 multiplications
  Mémoire : $\Omega(n^2)$ nombres de 32 bits
- utilisation du CRT selon Kawamura
  Complexité : environ n^2 multiplications
  Mémoire : $\Omega(n^2)$ nombres de 32 bits

Extension de base

Les étapes 1 et 3 sont en $O(n)$.
Comment effectuer les deux extensions de base ?
Quelle est leur complexité ?
Trois algorithmes possibles :

- utilisation de MRS (système en base mixte), une représentation associée à RNS
  Complexité : environ $\frac{3}{2}n^2$ multiplications
  Mémoire : $\Omega(n)$ (ou $\Omega(n^2)$) nombres de 32 bits
- utilisation du CRT selon Bajard
  Complexité : environ n^2 multiplications
  Mémoire : $\Omega(n^2)$ nombres de 32 bits
- utilisation du CRT selon Kawamura
  Complexité : environ n^2 multiplications
  Mémoire : $\Omega(n^2)$ nombres de 32 bits

Plan

Pseudo-nombres de Mersenne et opérations sur les canaux

Pour accélérer les calculs sur les canaux, on prend comme moduli m_i des pseudo-nombres de Mersenne, i.e. des nombres premiers de la forme :

$$m_i = 2^{32} - c_i$$

avec $c_i < 2^{15}$.

La réduction d'un nombre a de 64 bits se fait alors ainsi :

1: Écrire $a = a_1 2^{32} + a_0$ avec $0 \leq a_1, a_0 < 2^{32}$
2: $a' \leftarrow a_1 \times c_i + a_0$ $\{a' \equiv a\ [m]$ et $a' < 2^{48}\}$
3: Écrire $a' = a'_1 2^{32} + a'_0$ avec $0 \leq a'_1, a'_0 < 2^{32}$ $\{a'_1 < 2^{16}\}$
4: $a'' \leftarrow a'_1 \times c_i + a'_0$ $\{a'' \equiv a\ [m]$ et $a'' < 2^{32}\}$
5: **if** $a'' > m$ **then**
6: **return** $a'' - m$
7: **else**
8: **return** a''
9: **end if**

TABLE: Comparaison de la vitesse d'un déchiffrement RSA 1024 avec RNS (avec exponentiation par fenêtre glissante de 6 bits) et avec GMP, sur x86_64

		Par. MRS		Seq. MRS		Bajard		Kawamura	
Taille moduli	SSE	Durée (ms)	GMP ratio[a]	Durée (ms)	GMP ratio	Durée (ms)	GMP ratio	Durée (ms)	GMP ratio
32 bits	no	19.95 ± 0.02	21.82 ± 0.10	27.19 ± 0.03	30.76 ± 0.12	4.87 ± 0.00	5.53 ± 0.01	4.73 ± 0.00	5.36 ± 0.01
32 bits	2	18.15 ± 0.02	19.80 ± 0.04	26.91 ± 0.02	30.47 ± 0.07	4.60 ± 0.00	5.21 ± 0.01	4.46 ± 0.00	5.06 ± 0.01
32 bits	4.1	16.82 ± 0.01	18.98 ± 0.02	26.86 ± 0.03	30.47 ± 0.05	4.54 ± 0.00	5.13 ± 0.01	4.43 ± 0.00	5.02 ± 0.01
64 bits	n/a	7.35 ± 0.01	8.31 ± 0.02	9.17 ± 0.01	10.44 ± 0.03	2.74 ± 0.01	3.11 ± 0.01	2.57 ± 0.00	2.91 ± 0.01

TABLE: Comparaison de la vitesse d'un déchiffrement RSA 1024 avec RNS et avec la librairie multi-precision (exponentiation rapide), sur ARM7, à 7.3728 MHz

	Durée (ms)	Ratio avec librairie multi-precision
RNS MRS parallèle	24.89 $\pm$ 0.17	2.38
RNS MRS séquentiel	23.88 $\pm$ 0.19	2.28
RNS Bajard	11.81 $\pm$ 0.07	1.13
RNS Kawamura	11.56 $\pm$ 0.08	1.11
Multi-precision	10.46 $\pm$ 0.01	1
Multi-precision[a]	8.80 $\pm$ 0.01	0.84

[a] Avec boucles déroulées.

Instructions personnalisées pour Nios II

Nios II est un microprocesseur “soft-core” de type RISC supportant des instructions personnalisées à 3 opérandes.

Instructions personnalisées pour Nios II

Nios II est un microprocesseur "soft-core" de type RISC supportant des instructions personnalisées à 3 opérandes.

- Accumulateur de 72 bits avec une multiplication 32×32 bits avec résultat de 64 bits.

  Usage : multiplication, algorithmes de Bajard et Kawamura

- Unité « multiplication réduction » :

$$x \times (y - z) + t \mod m$$

avec x, y, z, t des entiers de 32 bits

tels que $y - z$ a seulement 16 bits et $m = 2^{32} - c$ est un pseudo nombre de Mersenne.

Usage : MRS séquentiel et réduction de nombres de 64 bits modulo m_i car

$$x \mod m = x_1 \times (c - 0) + x_0 \mod m_i.$$

si $x = x_1 2^{32} + x_0$ et x_0, x_1 sont des nombres de 32 bits.

Nios II benchmark

Ces deux unités sont implémentées en Verilog en utilisant seulement un multiplieur 16×32 bits.

TABLE: Comparaison de la vitesse d'un déchiffrement RSA 1024 sur ARM7 et sur Nios II (exponentiation rapide), à 7.3728 MHz

	Nios II (ms)	ARM (ms)	Ratio
RNS MRS parallèle	13.75 $\pm$ 0.09	24.89 $\pm$ 0.17	1.81
RNS MRS séquentiel	8.88 $\pm$ 0.05	23.88 $\pm$ 0.19	2.69
RNS Bajard	6.72 $\pm$ 0.19	11.81 $\pm$ 0.07	1.76
RNS Kawamura	6.39 $\pm$ 0.10	11.56 $\pm$ 0.08	1.81

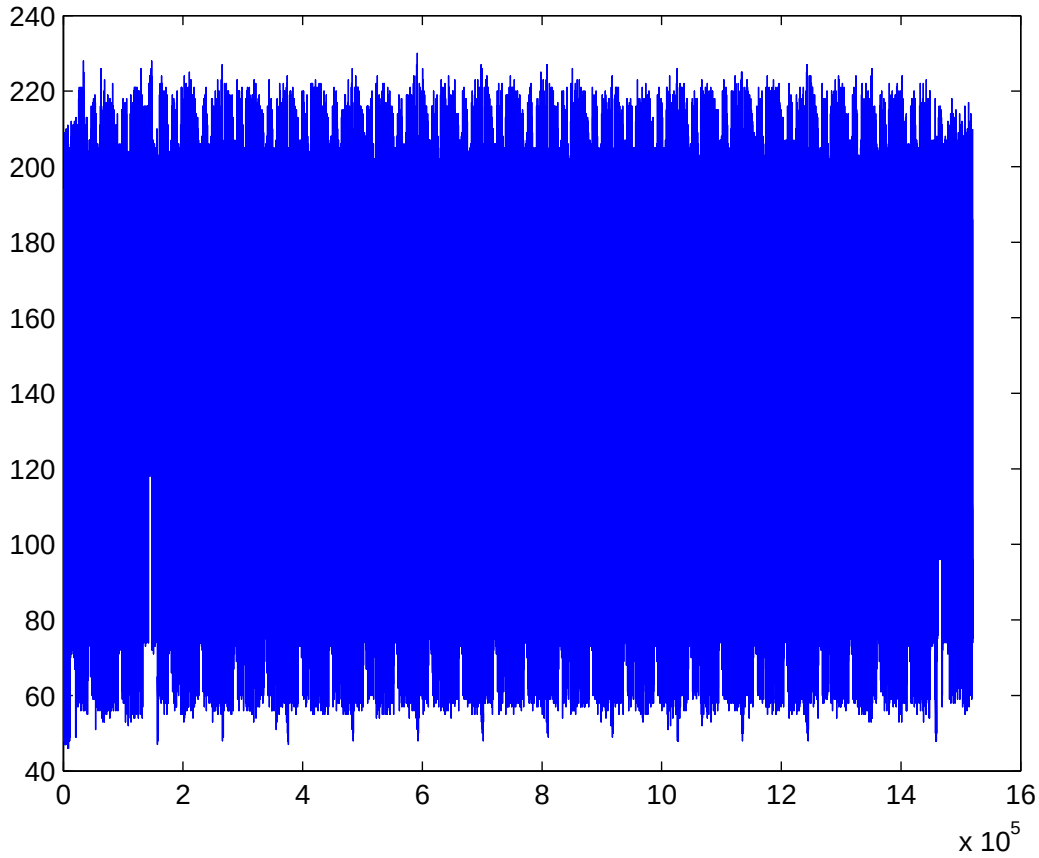
Plan

Analyse simple de consommation (SPA) and analyse différentielle de consommation (DPA)

La consommation électrique de tout microprocesseur dépend des instructions exécutées et des données traitées.
L'analyse de cette consommation peut permettre de retrouver la clé secrète utilisée lors d'un déchiffrement RSA.

Analyse simple de consommation (SPA) and analyse différentielle de consommation (DPA)

On appelle trace, le graphe de la consommation en fonction du temps lors d'un déchiffrement RSA.



Analyse simple de consommation (SPA) and analyse différentielle de consommation (DPA)

La consommation électrique de tout microprocesseur dépend des instructions exécutées et des données traitées.
L'analyse de cette consommation peut permettre de retrouver la clé secrète utilisée lors d'un déchiffrement RSA.

On appelle trace, le graphe de la consommation en fonction du temps lors d'un déchiffrement RSA.

Deux grands types d'attaques :

- SPA : interprétation directe d'une seule trace
- DPA : utilisation de plusieurs traces et de méthodes statistiques

Une attaque SPA simple

Dans certaines implémentations de RSA, la multiplication modulaire et la mise au carré ne consomment pas la même quantité d'électricité.

Il est alors possible de distinguer les multiplications des mises au carré, et de retrouver l'exposant.

d	1	0	1	1	0	1	0	1
M / S / F		F	S	M	S	M	S	S	M	S	S	M
Result	x	x^2	x^4	x^5	x^{10}	x^{11}	x^{22}	x^{44}	x^{45}	x^{90}	x^{180}	x^{181}

- F : multiplication par x et mise au carré
- S : mise au carré
- M : multiplication par x

SPA et notre implémentation RNS

Cette attaque est difficile avec RNS car multiplication et mise au carré sont identiques.

SPA et notre implémentation RNS

Cette attaque est difficile avec RNS car multiplication et mise au carré sont identiques.

De plus, pour se protéger contre d'autres attaques SPA simple, des versions *side-channel atomic* des algorithmes d'exponentiation sont utilisées :
la même séquence d'instruction est exécutée un certain nombre de fois par le microprocesseur
$\Rightarrow$ pas de sauts dépendant des données.

Une attaque DPA simple I

Objectif

Distinguer les multiplications par x des mises au carré, en cherchant les endroits où le premier canal x_1 de x est chargé dans un registre.

On effectue s déchiffrements RSA avec différents x : $x^{(1)}$, ..., $x^{(s)}$ (choisis aléatoirement). Et on enregistre les traces correspondantes.

Une attaque DPA simple I

Objectif

Distinguer les multiplications par x des mises au carré, en cherchant les endroits où le premier canal x_1 de x est chargé dans un registre.

On effectue s déchiffrements RSA avec différents x : $x^{(1)}, \ldots, x^{(s)}$ (choisis aléatoirement). Et on enregistre les traces correspondantes.

Représentons x comme un échantillon d'une variable aléatoire X et les traces comme une liste de variables aléatoires (T_t).

Une attaque DPA simple II

Soit W le premier canal de X et $H = HW(W)$.

Comme la consommation du microprocesseur ARM7TDMI-S dépend linéairement du poids de Hamming (HW) des données chargées à partir de la mémoire :

- si t correspond à un chargement du premier canal de X :

$$H = \mathrm{HW}(W) \text{ dépend linéairement de } T_t$$

- sinon :

$$H = \mathrm{HW}(W) \text{ est indépendant de } T_t$$

Une attaque DPA simple III

Comment distinguer ces deux cas ?

Une attaque DPA simple III

Comment distinguer ces deux cas ?

Il est possible d'utiliser le coefficient de corrélation linéaire de Pearson

$$\rho_t = \rho(H, T_t) = \frac{\mathrm{Cov}(H, T_t)}{\sqrt{\mathrm{Var}(H)\,\mathrm{Var}(T_t)}} \in [-1, 1].$$

En effet :

- $|\rho_t| = 1$ si H et T_t sont linéairement dépendant ;
- $\rho_t = 0$ si H et T_t sont indépendants.

Une attaque DPA simple (résultats)

d	1	0	1	1	0	1	0	1
M / S / F		F	S	M	S	M	S	S	M	S	S	M
Result	x	x^2	x^4	x^5	x^{10}	x^{11}	x^{22}	x^{44}	x^{45}	x^{90}	x^{180}	x^{181}

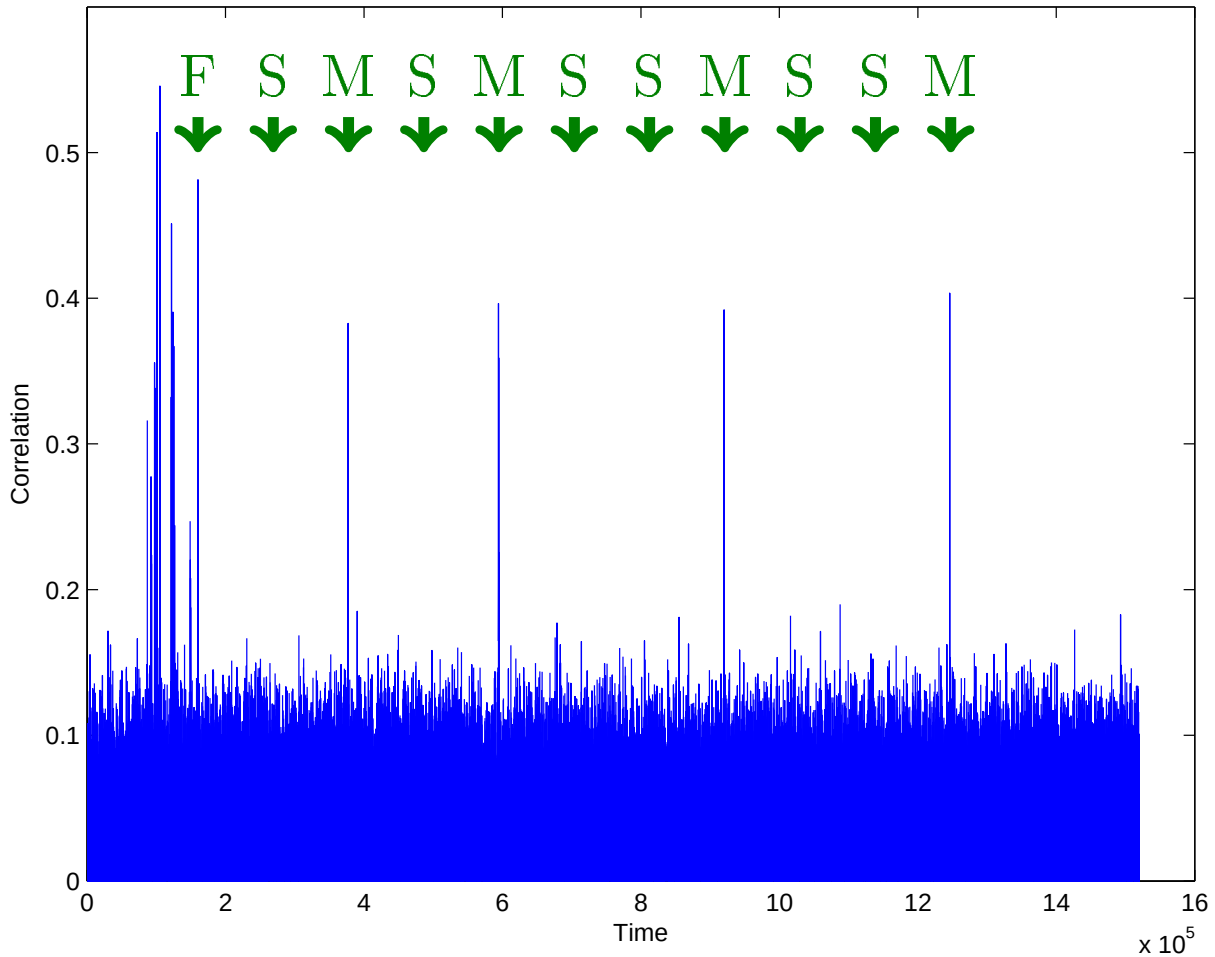


FIGURE: Corrélation avec 500 traces, $d = 181$ et $n = 4$

Randomisation intra-base

Une première idée pour rendre les attaques plus difficiles consiste à changer régulièrement l'ordre des moduli dans chacune des bases $\mathfrak{B}_1$ et $\mathfrak{B}_2$. Seul l'ordre des opérations est changé (et certaines constantes dans certains cas).

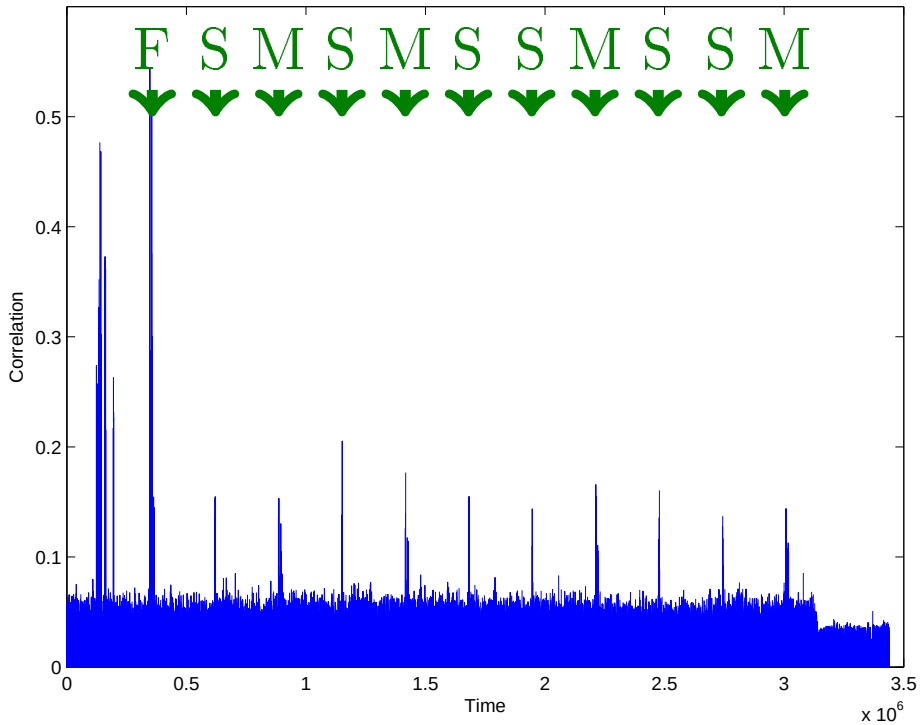
Randomisation intra-base

Une première idée pour rendre les attaques plus difficiles consiste à changer régulièrement l'ordre des moduli dans chacune des bases $\mathfrak{B}_1$ et $\mathfrak{B}_2$. Seul l'ordre des opérations est changé (et certaines constantes dans certains cas).

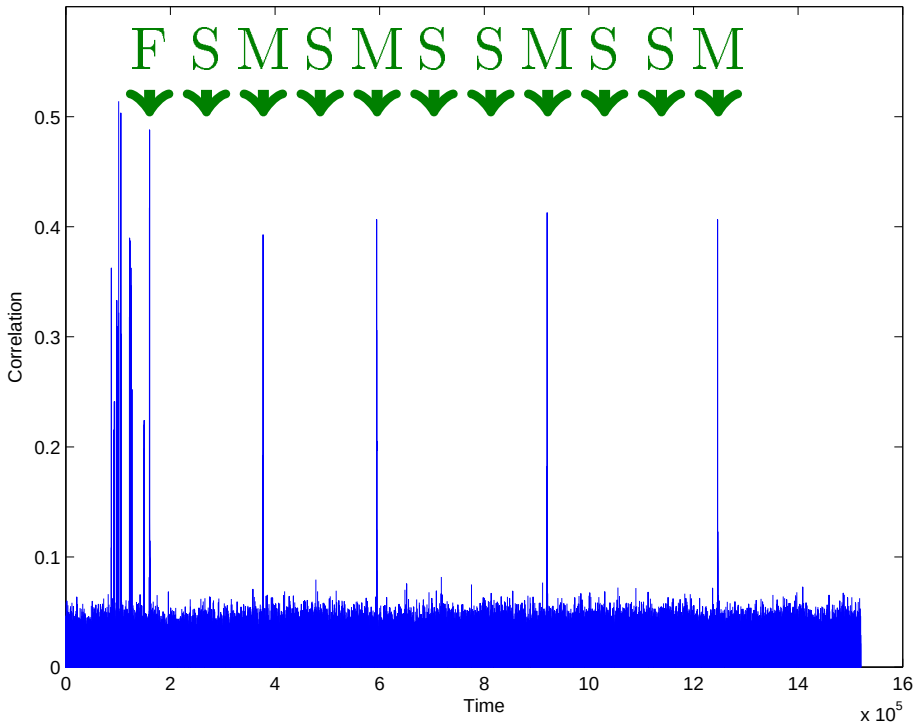
En théorie, cela divise le coefficient de Pearson par $n = 4$ et multiplie le nombre de traces nécessaires par $n^2 = 16$.

Il est cependant possible de ne multiplier le nombre de traces nécessaires que par $n = 4$.

Randomisation intra-base



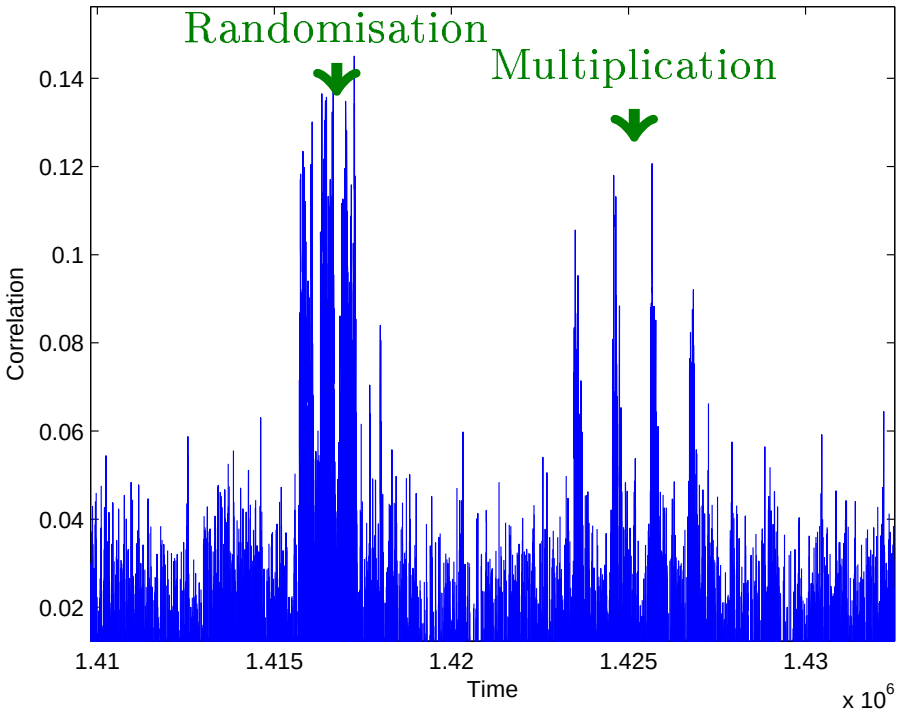
(a) Randomisation intra-base



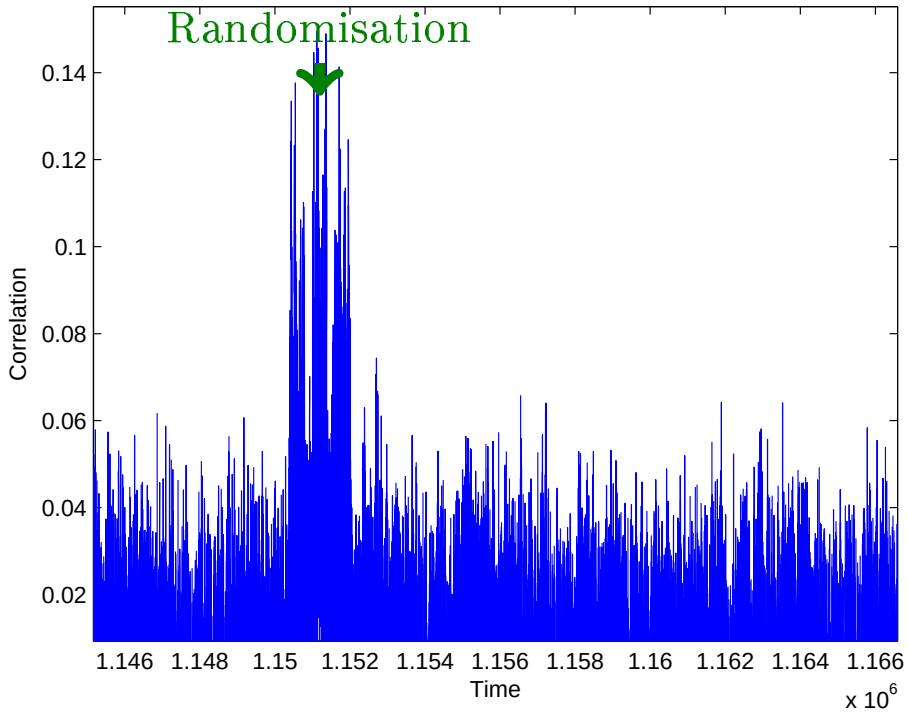
(b) Pas de randomisation

FIGURE: Corrélation avec 3000 traces

Randomisation intra-base



(a) Multiplication



(b) Carré

FIGURE: Corrélation avec 3000 traces (zoom trace randomisation)

Coût des contre-mesures sur ARM

	Pas de rand.	Intra-base[a]		Inter-base[b]	
Algorithm	Durée (ms)	Durée (ms)	Ratio	Durée (ms)	Ratio
Par. MRS	24.89	29.88[f] 28.26[g]	1.20[f] 1.14[g]	55.63[f]	2.24[f]
Seq. MRS	23.88	36.04[f] 35.61[g]	1.50[f] 1.50[g]	53.92[f]	2.26[f]
Kawamura	11.56	14.27[g]	1.23[g]	n/a	n/a

[a] Avant chaque multiplication modulaire.
[b] Avant 1 multiplication modulaire sur 5.
[f] La représentation des nombres est randomisées en mémoire.
[g] La représentation des nombres n'est pas randomisées en mémoire.

Conclusion — Contributions

- Théorie
  - Étude de la complexité de la réduction de Montgomery avec RNS
  - Analyse des attaques SPA et DPA connues
  - Analyse de la randomisation des bases (LRA)
  - Extension à la réduction classique de Montgomery
  - Analyse de l'arithmétique redondante de Montgomery (RMA)
- Pratique
  - Différentes de RNS RSA en C
  - Optimisation des parties critiques en assembleur pour x86_64 (+SSE), ARMv4 et Nios II
  - Proposition et implémentation d'une extension de jeu d'instructions pour Nios II
  - Attaques DPA réelles
- Divers
  - Étude de l'implémentation efficace des permutations
  - Nouvelle multiplication en temps constant pour ARMv4
  - Proposition d'un algorithme *side-channel atomic Montgomery ladder*

Conclusion — Travail futur

- Optimisation du programme : meilleur ordonnancement des instructions (x86_64), déroulement des boucles critiques, ...
- Meilleure comparaison avec le travail de Lim et Philips (une autre extension de jeu d'instructions)
- Analyse des attaques par *template DPA*
- Analyse des attaques par fautes (en particulier DFA)

Merci de votre attention

Contre-mesures classiques

Il existe deux contre-mesures classiques :

- masquage du message : changer la représentation de x utilisée pendant l'exponentiation, par exemple en xr^e (r aléatoire) car :

$$(xr^e)^d = x^d$$

- masquage de l'exposant : remplacer l'exposant d par $d + r\Phi(N)$ (r aléatoire) car :

$$x^{d+r\Phi(N)} = x^d$$

Rappels sur RSA :

- $(x^e)^d = x$
- $x^{\Phi(N)} = 1$

Corrélation interne

Pour contrer le masquage du message, il est possible de choisir $H = T_{t'}$ avec t' l'instant où x_1 est chargé lors de la première multiplication.

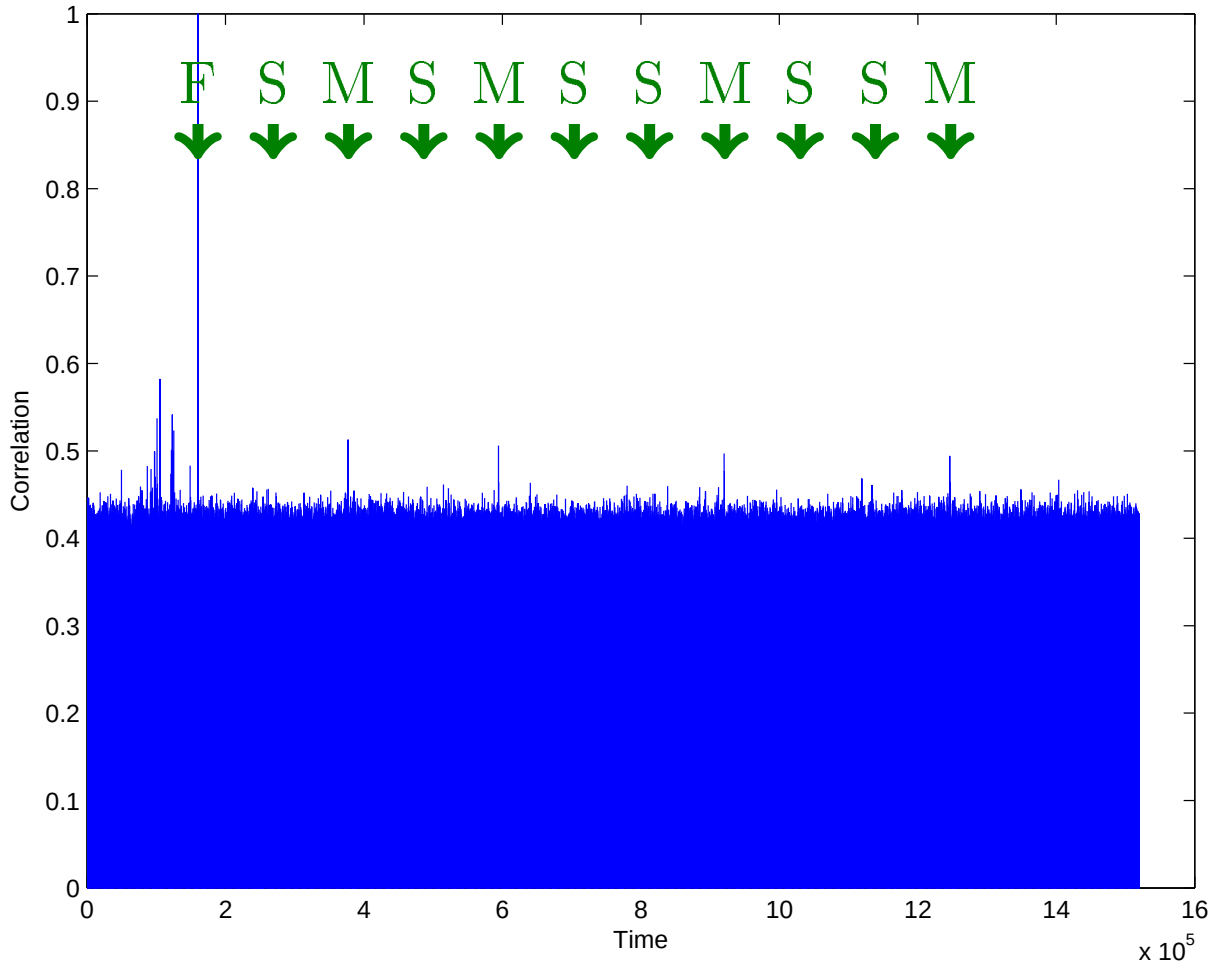


FIGURE: Corrélation avec 500 traces, $d = 181$ et $n = 4$

Randomisation inter-base et arithmétique redondante de Montgomery (RMA)

- randomisation inter-base : permutation aléatoire de tous les moduli ensemble. Comme $R = M_1$, les représentations de Montgomery des nombres utilisés changent.
- arithmétique redondante de Montgomery (RMA) : ajout d'un multiple rN (r aléatoire) aux nombres utilisés.

Randomisation inter-base et arithmétique redondante de Montgomery (RMA)

- randomisation inter-base : permutation aléatoire de tous les moduli ensemble. Comme $R = M_1$, les représentations de Montgomery des nombres utilisés changent.
- arithmétique redondante de Montgomery (RMA) : ajout d'un multiple rN (r aléatoire) aux nombres utilisés.

Remarques :

- Ces contre-mesures offrent plus de possibilité que le masquage des messages (contre-mesures classiques).
- RMA est très peu coûteux, contrairement à la randomisation inter-base qui nécessite deux multiplications de Montgomery par variable intermédiaire.
- RMA utilisé avec la randomisation intra-base apporte approximativement les mêmes avantages que la randomisation inter-base.