UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS
Laboratoire d'informatique avancée de Saint-Denis
Équipe PASTIS

Trustable Homomorphic Computation

3 juin 2021

SOIF — séance 3

Pablo Rauzy <pr@up8.edu>
pablo.rauzy.name

SOIF — séance 3

Overview

- Protecting CRT-RSA against fault injection attacks.
- Generalizing modular extension.
- Trustable homomorphic computation.

Background in Implementation Security

- RSA, CRT-RSA, and the BellCoRe attack.
- Modular extension.

RSA (Rivest, Shamir, Adleman) — **Definition**

RSA is an algorithm for public key cryptography. It can be used as both an encryption and a signature algorithm.

- Let M be the message,
  (N, e) the public key, and
  (N, d) the private key,
  such that $d \cdot e \equiv 1 \mod \varphi(N)$.
- The signature S is computed by $S \equiv M^d \mod N$.
- The signature can be verified by checking that $M \equiv S^e \mod N$.

CRT (Chinese Remainder Theorem) — Definition

CRT-RSA is an optimization of the RSA computation which allows a fourfold speedup.

- Let p and q be the primes from the key generation ($N = p \cdot q$).
- These values are pre-computed (considered part of the private key):
  - $d_p \doteq d \bmod (p-1)$
  - $d_q \doteq d \bmod (q-1)$
  - $i_q \doteq q^{-1} \bmod p$
- S is then computed as follows:
  - $S_p = M^{d_p} \bmod p$
  - $S_q = M^{d_q} \bmod q$
  - $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p)$
    (Garner recombination).

BellCoRe (Bell Communications Research) — **Definition**

The BellCoRe attack consists in revealing the secret primes p and q by faulting the computation. It is very powerful as it works even with very random faulting.

- If S_p (resp. S_q) is faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), the attacker:
  - gets an erroneous signature $\widehat{S}$,
  - can recover p (resp. q) as $\gcd(N, S - \widehat{S})$.

Why does it Work?

- For all integer x, $\gcd(N, x)$ can only take 4 values:
  - 1, if N and x are co-prime,
  - p, if x is a multiple of p,
  - q, if x is a multiple of q,
  - N, if x is a multiple of both p and q, i.e., of N.

- If S_p is faulted (i.e., replaced by $\widehat{S_p} \neq S_p$):
  - $S - \widehat{S} = q \cdot \left((i_q \cdot (S_p - S_q) \mod p) - (i_q \cdot (\widehat{S_p} - S_q) \mod p)\right)$

  $\Rightarrow \gcd(N, S - \widehat{S}) = q$

- If S_q is faulted (i.e., replaced by $\widehat{S_q} \neq S_q$):
  - $S - \widehat{S} \equiv (S_q - \widehat{S_q}) - (q \mod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \mod p$

  $\Rightarrow \gcd(N, S - \widehat{S}) = p$

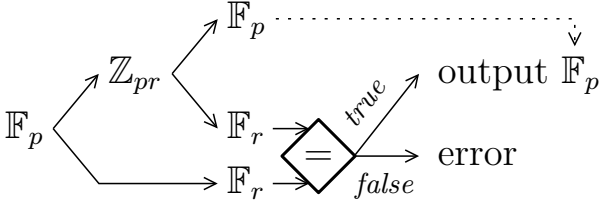
- Many countermeasures have been proposed:
  - ∼20 papers,
  - from 1999 to now,
  - both from academia and industry.
- Including:
  - Shamir (1999),
  - Aumüller et al. (2002),
  - Vigilant (2008) + Coron et al. (2010).

- Inputs:
  - a high-level description of the algorithm,
  - an attack success condition,
  - a fault model.
- Output:
  - the list of working attacks, or
  - a proof that the computation is resistant to fault injections.
- `https://pablo.rauzy.name/sensi/finja.html`

Classification

Countermeasure	Family	Verification method/count	Order intended	Order actual	Small subrings usage
Shamir (1999)	Shamir	test / 1	1	0	$r_1 = r_2$, consistency of intermediate signatures
Joye et al. (2001)	Shamir	test / 2	1	0	checksums of the intermediate CRT signatures
Aumüller et al. (2002)	Shamir	test / 5	1	1	$r_1 = r_2$, consistency of the checksums of both intermediate signatures
Blömer et al. (2003)	Shamir	infection / 2	1	1	direct verification of the intermediate CRT signatures, CRT recombination happens in overring
Ciet & Joye (2005)	Shamir	infection / 2	2	1	checksums of the intermediate CRT signatures, CRT recombination happens in overring
Giraud (2006)	Giraud	test / 1	1	1	NA
Boscher et al. (2007)	Giraud	test / 1	1	1	NA
Vigilant (2008)	Shamir	test / 7	1	1	$r_1 = r_2$, embedded control values, CRT recombination happens in overring
Rivain (2009)	Giraud	test / 2	1	1	NA
Kim et al. (2011)	Giraud	infection / 6	1	1	NA

- Many of these countermeasures are patented.
- Most of them are doing the exact same thing: modular extension.
- The idea is to use the isomorphism between $\mathbb{F}_p \times \mathbb{F}_r$ and $\mathbb{Z}_{pr}$.



Notation: $\mathbb{F}_p$ is a shortcut for $\mathbb{Z}/p\mathbb{Z}$

Generalizing Modular Extension

- Modular extension is not tied to RSA.
- Automation and application to elliptic curve cryptography.

- The working factors of modular extension based countermeasures:
  - are not tied to the BellCoRe attack,
  - nor to the CRT-RSA algorithm.
- Cost-effective integrity verification of any arithmetic computation.

Inversion in Direct Products

Divisions optimization — Proposition

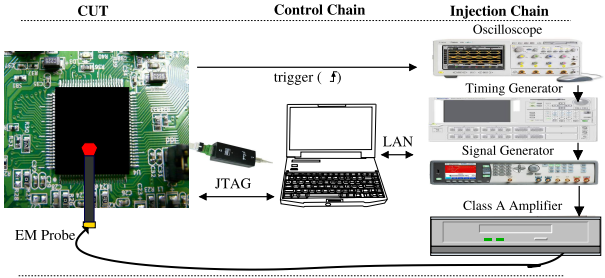
To get the inverse of z in $\mathbb{F}_p$ while computing in $\mathbb{Z}_{pr}$, one has:

- $z = 0 \bmod r \implies (z^{p-2} \bmod pr) \equiv z^{-1} \mod p$,
- otherwise $(z^{-1} \bmod pr) \equiv z^{-1} \mod p$.

proof sketch:

- If $z = 0 \mod r$, then z is not invertible in $\mathbb{Z}_{pr}$:
  - but z^{p-2} exists in $\mathbb{Z}_{pr}$,
  - and $(z^{p-2} \bmod pr) \bmod p = z^{p-2} \bmod p = z^{-1} \bmod p$.

- Inputs:
  - an asymmetric cryptography algorithm to be protected,
  - a desired redundancy level.
- Output:
  - the (proved to be the) same algorithm
  - provably protected against fault injection attacks.
- `https://pablo.rauzy.name/sensi/enredo.html`



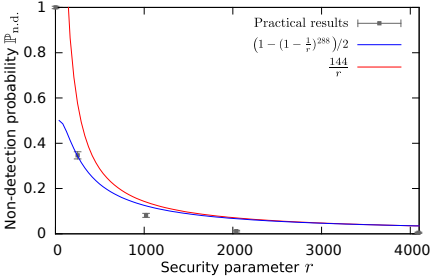
Field characteristic	p =	0xffffffffffffffffffffffffffffffffeffffffffffffff
Curve equation coefficients	a =	0xffffffffffffffffffffffffffffffffeffffffffffffc
	b =	0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1
Point coordinates	Gx =	0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012
	Gy =	0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811
Point order	ord =	0xffffffffffffffffffffff99def836146bc9b1b4d22831

Parameters of our ECSM implementation (namely NIST P-192)

Security Results

r value	r size (bit)	Positives (%) true	Positives (%) false	Negatives (%) true	Negatives (%) false
1	1	0.00	0.00	2.74	97.26
251	8	63.65	0.00	2.56	33.79
1021	10	89.09	0.00	2.96	7.95
2039	11	98.82	0.00	0.00	1.18
4093	12	97.61	0.00	1.91	0.48
65521	16	97.79	0.00	2.21	0.00
4294967291	32	97.19	0.00	2.81	0.00
18446744073709551557	64	99.79	0.00	0.21	0.00

≈ 1000 tests for each value of r



Performance Results

r value	r size (bit)	time (ms) $\mathbb{Z}_{pr}$	time (ms) $\mathbb{F}_r$	test	overhead
1	1	683	24	$\ll 1$	$\times 1.04$
251	8	883	91	$\ll 1$	$\times 1.43$
1021	10	899	100	$\ll 1$	$\times 1.46$
2039	11	902	197	$\ll 1$	$\times 1.61$
4093	12	903	197	$\ll 1$	$\times 1.61$
65521	16	883	189	$\ll 1$	$\times 1.56$
4294967291	32	832	172	$\ll 1$	$\times 1.47$
18446744073709551557	64	996	246	$\ll 1$	$\times 1.82$

Signature verification overhead $\approx \times 4.5$.

Code C + **`mini-gmp`** compiled with **`gcc -O0`** (no optimization).

Modular Extension is Parallel

- The computation in $\mathbb{F}_r$ and in $\mathbb{Z}_{pr}$ are completely independent.
- They can be executed in any order and in particular on different machines.

Trustable Homomorphic Computation

- Homomorphic cryptography, privacy concerns.
- Related works.
- Leveraging modular extension.
- THC.

Homomorphic cryptosystem — **Definition**

A cryptosystem is homomorphic if it satisfies algebraic properties allowing to carry out operations on ciphertexts.
Let $F : A \to B$ an encryption function, let $\odot_A$ and $\odot_B$ operations on A and B, (F, F^{-1}) is homomorphic for $\odot_A$ if $\odot_B$ is such that $F^{-1}(F(x) \odot_B F(y)) = x \odot_A y$.

- Allows to delegate computation on privacy-sensitive data...
- ... provided that we trust the third-party with the computation.

Trivial Example

- An insurance company offers multiple plans for weight-related diseases.
- It also provides a free service for computing one's body mass index.

$$BMI = \frac{\text{mass}}{\text{height}^2}$$

Privacy Concerns

- People do not want to send their personal information in clear over the network.
- They also do not want to reveal them to the insurance company.
- Thus the BMI computation service uses homomorphic cryptography:
  - the user do not send $m = \text{mass}$ and $h = \text{height}$,
  - instead, they send $\mathcal{E}(m)$ and $\mathcal{E}(h)$, homomorphically encrypted values.

Trust

- The user must now trust the insurance company with computing

$$\mathcal{E}(BMI) = \frac{\mathcal{E}(m)}{\mathcal{E}(h) \times_{\mathcal{E}} \mathcal{E}(h)}\varepsilon.$$

- While the insurance company would benefit from computing instead

$$\mathcal{E}(BMI') = \frac{\mathcal{E}(m) +_{\mathcal{E}} \mathcal{E}(20)}{\mathcal{E}(h) \times_{\mathcal{E}} \mathcal{E}(h)}\varepsilon.$$

- How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption
  Parno et al. 2011 (IACR ePrint 2011/597).
- Pinocchio: Nearly Practical Verifiable Computation
  Parno et al. 2013 (IACR ePrint 2013/279).
- Efficiently Verifiable Computation on Encrypted Data
  Fiore et al. 2014 (IACR ePrint 2014/202).
- Verifiable Computation on Outsourced Encrypted Data
  Lai et al. 2014.

Problems

- Introduce new complicated cryptographic constructions (e.g., “homomorphic encrypted authenticator” in Lai et al.).
- Stay at a theoretical level: only implemented in LaTeX.
- No actual implementation and novelty means no real tests:
  - Security flaws?
  - Complexity?
  - Practical feasibility of implementation?
- Require cooperation of the third-party.

Leveraging Modular Extension

- Modular extension is practical, proved, and formally studied.
- Let's try to apply it to homomorphic cryptography!

Project

- Getting to know better the field of homomorphic cryptography.
- Finding (or inventing?) a (somewhat) fully homomorphic cryptosystem which would support modular extension.
  - No tests depending on the modular values.
- Finding (or writing?) an hackable implementation of it.
  - YASHE from Bos et al. 2013 (IACR ePrint 2013 2013/075)?
- Prototyping the verification of delegated computation using modular extension.