# Protocols, cryptanalysis and mathematical cryptology (INFO-F514)

## “Provable Security”

Christophe Petit

Université libre de Bruxelles

## Sensitization campaigns

1. Together against sexual harassment and violences
   `https://www.ulb.be/fr/sante-et-bien-etre/si-cest-pas-oui-cest-non`

2. COVID-19: please go get vaccinated !

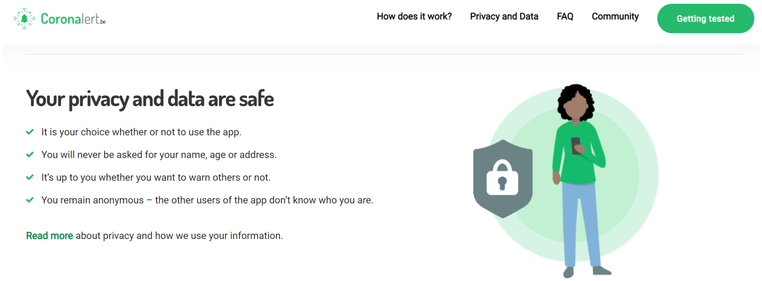
# Protocols, cryptanalysis and mathematical cryptology (INFO-F514)

## “Provable Security”

Christophe Petit

Université libre de Bruxelles

## Proving security?



Your privacy and data are safe

- ✓ It is your choice whether or not to use the app.
- ✓ You will never be asked for your name, age or address.
- ✓ It's up to you whether you want to warn others or not.
- ✓ You remain anonymous – the other users of the app don't know who you are.

Read more about privacy and how we use your information.

## Outline

“Provable security”: reductionist approach and hard problems

Example: ElGamal Encryption protocol

Limitations to the provable security approach

# Outline

“Provable security”: reductionist approach and hard problems

Example: ElGamal Encryption protocol

Limitations to the provable security approach

## Proving security?

- How can we test security?
- No matter what kind of experiments you run, the adversary might do something unexpected
- Need to anticipate all attacks in advance
- In cryptography, best you can do:
  - Make precise definitions of security
  - Argue that your protocol satisfies those definitions
- Warning: always beware what is meant by “secure”

## Reductionist approach

- Precisely define what it means to break the protocol
  - Adversary's goal
  - Adversary's resources
  - Adversary's access to the system
- Choose your favorite hard problem
  - A computational problem that cannot be solved, even by clever people with the best computers available
- Build a protocol so that you can prove

  Breaking the protocol $\Rightarrow$ Solving the hard problem

## Security guarantees, ideally

Attack on security protocol
(TLS, SSH, Signal, evoting, etc)

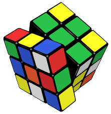
$$\Downarrow$$

Attack on one of the cryptographic building blocks
(RSA signatures, Diffie-Hellman key agreement, etc)

$$\Downarrow$$

Algorithm for some hard computational problems
(factoring, computing discrete logarithms, etc)

## On the meaning of "hard"

- Is  hard ?
- Is adding two integers hard?
- Is multiplying two integers hard?
- Is inverting a matrix hard? what if it has billions of rows and columns?
- Is factoring integers hard? what about 15?

## On the meaning of “hard”

- **Asymptotic hardness:** no polynomial-time adversary can solve a given problem instance with a non-negligible probability
  - Choose a security parameter $\lambda \sim$ problem size
  - Adversary is an algorithm
  - Adversary’s complexity is a function of $\lambda$
  - Negligible $=$ smaller than $1/p(\lambda)$ for any polynomial $p$
- **Concrete hardness:** cannot solve the problem today, even with the best computers available
  - Example: cannot succeed with probability bigger than $2^{-50}$ in time less than $2^{80}$ seconds using less than $2^{40}$GB

## Public Key Cryptography specificities

- Public key hard problems often from Number Theory or other Mathematics areas
- Well-suited for asymptotic hardness and “proofs” (security is a function of problem size)
- Asymptotic hardness *evidenced* by long research history
- Concrete hardness evaluated by collective accumulation of knowledge, challenges, benchmarks

## Integer factorization

- Every integer can be uniquely decomposed as a product of prime numbers (up to permutations)
- Given prime numbers $p_i$, easy to compute their product
- Given $n = pq$ where $p, q$ are well-chosen primes, best known algorithm requires

$$L_n\left(\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right) = \exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\log n)^{1/3}(\log\log n)^{2/3}\right)$$

operations to factor $n$

- Biggest RSA challenge solved today has 829 bits

## Computing discrete logarithms

- Let $p$ be a prime. The function

$$f : [0, \ldots, p-2] \to \mathbb{Z}_p : x \to g^x \bmod p$$

is easy to compute using square-and-mulitiply algorithm

```
1: Let $x = \sum_{i=0}^{n} x_i 2^i$
2: $a' \leftarrow a$; $c \leftarrow a^{x_0}$;
3: for i=1 to n do
4:     $a' \leftarrow a'^2 \bmod p$
5:     if $x_i = 1$ then
6:         $c \leftarrow ca' \bmod p$
7:     end if
8: end for
9: return $c$
```

# Computing discrete logarithms

- Let $p$ be a prime. The function

$$f : [0, \ldots, p-2] \rightarrow \mathbb{Z}_p : x \rightarrow g^x \bmod p$$

  is easy to compute using square-and-mulitiply algorithm
- Computing the inverse function is believed to be hard, called the discrete logarithm problem
- Best algorithm today has a complexity

$$L_p\left(\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right) = \exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\log p)^{1/3}(\log\log p)^{2/3}\right)$$

- Current record has 768 bits (June 2016)

## Computing discrete logarithms (2)

- Discrete logarithm problem can be generalized to any finite cyclic group
- Groups used in cryptography
  - Multiplicative groups of finite fields, particularly $\mathbb{F}_p$; $\mathbb{F}_{p^n}$ used in pairing applications; $\mathbb{F}_{2^n}$ now to be avoided
  - Elliptic curve and hyperelliptic curve subgroups
- Complexity $O(|G|^{1/2})$ for well-chosen elliptic curves today

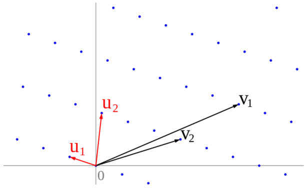
## Solving polynomial systems

- Let $K$ be a (finite) field, let $R = K[x_1, \ldots, x_n]$ be a polynomial ring over $K$, and let $f_i \in R$
- Solving polynomial systems

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \ldots \\ f_m(x_1, \ldots, x_n) = 0 \end{cases}$$

  is a hard problem in general, in fact it is NP-hard
- Used in some (still under scrutiny) cryptosystems
- More parameters, hence somewhat harder to evaluate

# Lattice problems



- A lattice is a discrete subgroup of $\mathbb{R}^n$
- SVP: Given a lattice, compute its shortest vector
- CVP: Given a lattice and an extra point, compute lattice vector closest to the point

- Active research area, now moving towards practice
- Many parameters involved, hardness evaluation underway

## Choosing your favorite hard problem

- Must be really hard
- Not just for you, after trying for 30minutes
- Ideally should have some history
- Must allow the construction of many cool protocols
- Those protocols must be efficient: small keys, fast encryption/decryption/signature, etc

# Outline

## Reductionist approach

- Precisely define what it means to break the protocol
  - Adversary's goal
  - Adversary's resources
  - Adversary's access to the system
- Choose your favorite hard problem
  - A computational problem that cannot be solved, even with the best computers available
- Build a protocol so that you can prove

$$\text{Breaking the protocol} \Rightarrow \text{Solving the hard problem}$$

## Outline

## What is secure encryption?

- Encryption and decryption algorithms
- Correctness
- Security properties:
  - Cannot find the key
  - Cannot decrypt a ciphertext
  - Cannot guess any information about a ciphertext
  - Even with a decryption oracle on other ciphertexts
  - . . .

## Public Key Encryption Scheme

- Key Generation algorithm **KeyGen**
  - Given security parameter $\lambda$
  - Return pair of private, public keys $(SK, PK)$
- Encryption algorithm **Enc**
  - Given public key $PK$ and message $m$
  - Return ciphertext $c$
- Decryption algorithm **Dec**
  - Given private key $SK$ and ciphertext $c$
  - Return message $m$

## Remarks

- **KeyGen**, **Enc** and **Dec** are probabilistic algorithms
- Keys implicitly define message and encryption space
- Algorithms must abort on wrong inputs
  (or better for **Dec**: return a random plaintext)
- In practice (for example using PGP)
  - Generate your own public, private key pair
  - Make your public key public
  - Everybody can encrypt using your public key
  - Only you can decrypt using your secret key

## Correctness

- A public key encryption scheme must be correct

$$\Pr\left[ m' = m \;:\; \begin{array}{l} (PK, SK) \leftarrow \textbf{KeyGen}(\lambda), \\ c \leftarrow \textbf{Enc}(PK, m), \\ m' = \textbf{Dec}(SK, c) \end{array} \right] \approx 1$$

(here $p \approx 1$ means $\epsilon := 1 - p$ is negligible, more precisely $\epsilon$ is a negligible function of the security parameter $\lambda$)

- Perfect correctness requires probability exactly 1

## Modeling security

- Interactive game between a good guy (the challenger) and a bad guy (the adversary)
- Scheme is *secure* if adversary cannot win the game except with a negligible advantage over trivial strategy
- One can consider various types of games corresponding to various types of adversaries
- Most popular security notions for public key encryption: IND-CPA, IND-CCA, IND-CCA2

## IND-CPA security

- Indistinguishability against chosen plaintext attacks
- Security game
  - Challenger runs $(PK, SK) \leftarrow \mathbf{KeyGen}(\lambda)$
  - Challenger sends $PK$ to adversary
  - Adversary chooses two messages $m_0, m_1$
  - Adversary sends $m_0, m_1$ to challenger
  - Challenger picks a random bit $b \leftarrow \{0, 1\}$
  - Challenger sends $c = \mathbf{Enc}(PK, m_b)$ to adversary
  - Adversary sends a bit $b'$ to challenger
  - Adversary wins if $b' = b$
- An encryption scheme is IND-CPA secure if

$$|\Pr[\text{Adversary wins}] - 1/2| \approx 0$$

## Deterministic encryption $\Rightarrow$ not IND-CPA secure

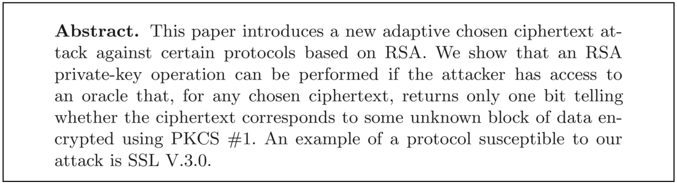
- No deterministic public key encryption scheme is IND-CCA secure
- Adversary can just compute $c_i = \mathbf{Enc}(PK, m_i)$ and compare with $c$ to always guess $b$ correctly
- Note that textbook RSA is deterministic, but RSA-OAEP variant used in practice adds some randomness to it

## IND-CCA(2) security

- IND-CCA(2) = indistinguishability against (adaptively) chosen ciphertext attacks
- Adversary is given an additional power: decryption oracle
  - Before sending $m_0, m_1$, adversary can call a decryption oracle on ciphertexts of its choice
  - In adaptive version, can also call the decryption oracle after receiving $c$ and before sending $b'$
    (but adversary not allowed to call oracle on $c$)
- Do we care in practice?

# Why caring about IND-CCA(2) security?

- Why in practice would you help the adversary by answering decryption queries?
- Bleichenbacher (Crypto'98): some protocols do, or not quite, but what they return suffices for attacks



**Abstract.** This paper introduces a new adaptive chosen ciphertext attack against certain protocols based on RSA. We show that an RSA private-key operation can be performed if the attacker has access to an oracle that, for any chosen ciphertext, returns only one bit telling whether the ciphertext corresponds to some unknown block of data encrypted using PKCS #1. An example of a protocol susceptible to our attack is SSL V.3.0.

## IND-CCA(2) vs IND-CPA security

- An attacker against IND-CPA game is also an attacker against the IND-CCA game, which is also an attacker against the IND-CCA2 game
- Hence

  $$\text{IND-CCA2 security} \Rightarrow \text{IND-CCA security}$$

  and

  $$\text{IND-CCA security} \Rightarrow \text{IND-CPA security}$$

- Converse not true, but IND-CCA(2) schemes usually built from IND-CPA schemes and hash functions

## Outline

## Discrete logarithms and related problems

- Let $G$ be a finite cyclic group of prime order $q$, and let $g$ be a generator
- Discrete logarithm problem: given $G$, $g$ and $h$, find $x$ such that $h = g^x$
- Diffie-Hellman problem: given $G$, $g$, $g^a$, $g^b$, compute $g^{ab}$
- Decisional Diffie-Hellman problem: let $G$, $g$, $g^a$, $g^b$, $g^c$ where $c$ is either $ab$ mod $q$ or random (with equal probability), guess correct distribution with probability significantly bigger than $1/2$

## Discrete logarithms and related problems (2)

- Efficient DLP algorithm implies efficient DHP algorithm, which implies efficient DDHP algorithm
- Converse implications not known, partly not expected
  - There are some groups where we know DDHP easy but we believe DHP hard
  - Hardness of DHP and DLP often equivalent in practice

## Key agreement

- Alice and Bob want to agree on a common secret key
- They only exchange public messages
- Eve can see all messages exchanged, yet she should not be able to infer the secret key

## Diffie-Hellman key agreement

- Choose $g$ generating a cyclic group $G$
- Alice picks a random $a$ and sends $g^a$
- Bob picks a random $b$ and sends $g^b$
- Alice computes $(g^b)^a = g^{ab}$
- Bob computes $(g^a)^b = g^{ab}$
- Eve cannot compute $a$, $b$ or $g^{ab}$ from $g^a$ and $g^b$ (discrete logarithm, Diffie-Hellman problems)

## ElGamal encryption

- Key generation:
  - Generate cyclic group $G$ of prime order $q$
  - Generate random $x \in [1, q]$
  - Pick a generator $g$ and set $h = g^x$
  - Private key is $x$ and public key is $(G, g, h)$
- Encryption: given public key $(G, g, h)$ and message $m$
  - Pick random $r \in [1, q]$
  - Return $c = (c_1, c_2) = (mh^r, g^r)$
- Decryption: given private key $x$ and ciphertext $c$
  - Return $m' = c_1 c_2^{-x}$

## ElGamal security

- Perfectly correct
- Computing private key from public key $\Leftrightarrow$ DLP
- Algorithm to decrypt messages $\Leftrightarrow$ DHP algorithm
  - Let $G, g, g^a, g^b$ a given DHP instance
  - Set $h = g^a$, $c_1 = 1$, $c_2 = g^b$, $c = (c_1, c_2)$
  - Ask decryption algorithm for $m = \mathbf{Dec}(c, SK)$
  - Return $g^{ab} = m^{-1}$
  - Conversely, let $m = c_1(g^{ab})^{-1}$ where $g^{ab}$ is a solution to DHP instance $G, g, h, c_2$

## ElGamal security (2)

- DDHP hard $\Rightarrow$ ElGamal IND-CPA secure
- Proof uses IND-CPA adversary to build DDHP solver
  - Let $G, g, g^a, g^b, g^c$ a given DDHP instance
  - Use $h = g^a$ as public key
  - Receive $m_1, m_2$ from adversary
  - Choose random bit $b$
  - Send $c = (g^b, m_b g^c)$ to adversary
  - Receive a guess $b'$ from adversary
  - Guess "$c = ab$" if $b = b'$

## ElGamal security (3)

- Suppose IND-CPA adversary wins with probability $1/2+\epsilon$
  - When $c=ab$, IND-CPA adversary receives inputs with expected distribution, so $b=b'$ with probability $1/2+\epsilon$
  - When $c$ random, his inputs are random and independent of $b$, so $b=b'$ with probability $1/2$
  - As both cases are as likely to occur, we have $b=b'$ with probability $1/2+\epsilon/2$
- El Gamal is not IND-CCA secure
  - Given $(c_1,c_2)=(m_b h^r, g^r)$, use decryption oracle on valid and different ciphertext $(c_1 h, c_2 g)$ to get $m$
- CCA-secure extension: Cramer-Shoup

# Outline

## Remember: “Provable Security”

- Precisely define what it means to break the protocol
  - Adversary’s goal
  - Adversary’s resources
  - Adversary’s access to the system
- Choose your favorite hard problem
  - A computational problem that cannot be solved, even by clever people with the best computers available
- Build a protocol so that you can prove

  Breaking the protocol $\Rightarrow$ Solving the hard problem

## Limits in Provable Security

- Remember "provable security"

  Breaking the protocol $\Rightarrow$ Solving the hard problem

- How can this go wrong?
  - Find out that security proof is wrong
  - Solve the underlying hard problem
  - Find weaknesses not covered by the proof

## Some security proofs are wrong

- With security proofs, evil is often in the details; many security proofs were later shown to be flawed
- It also happened to highly cited, peer-reviewed papers published at top conferences
- See some examples in Koblitz-Menezes, *Critical perspectives on provable security: Fifteen years of "another look at" papers*

## Not all "hard problems" are hard

- Many cryptographic protocols in the literature do not rely on the main problems in an area (DLP, factoring) but on *variants* or *subclasses* of these problems
- Typically: extra structure in subclass or variant useful for efficiency, functionality, or "provable security"
- Issue: do we have the same confidence in
    - DLP, factoring?
    - DDH, CDH, the RSA assumption?
    - One-Sided Modified SSCDH problem ?
- See chapter on cryptanalysis, and some examples in Koblitz-Menezes, *Critical perspectives on provable security: Fifteen years of "another look at" papers*

## Weaknesses not covered in the proof

- Easiest way to break security of a cryptographic protocol
- Security proof makes some assumptions $\pm$ explicitly
  - True randomness
  - Group cyclic, of large prime order
  - Black box access to the protocols
  - etc
- When these assumptions are not satisfied, no security guarantee is provided!

# Outline

"Provable security": reductionist approach and hard problems

Example: ElGamal Encryption protocol

Limitations to the provable security approach
- Subgroup attacks
- Weak randomness
- Strong adversaries

# Outline

## Diffie-Hellman key agreement

- Alice and Bob first agree on $p, g$

- Alice picks a random $a$ and sends $g^a \bmod p$
- Bob picks a random $b$ and sends $g^b \bmod p$

- Alice computes $(g^b)^a \bmod p = g^{ab} \bmod p$
- Bob computes $(g^a)^b \bmod p = g^{ab} \bmod p$

- Eve cannot compute $a$, $b$ or $g^{ab} \bmod p$ from $g^a \bmod p$ and $g^b \bmod p$ unless she solves a discrete logarithm or a computational Diffie-Hellman problem

## A naive implementation

```
p = random_prime(2**1024)
g = 2
a = random_integer(2**1024)
h = modexp(g,a,p)
```

- Is this secure?

## Pohlig-Hellman attack

```
p = random_prime(2**1024)
g = 2
a = random_integer(2**1024)
h = modexp(g,a,p)
```

- Let $N|p-1$ be the order of $g$, and $q$ a small divisor of $N$
- Then $(g^{N/q})$ has order $q$ and $(g^{N/q})^a = (h^{N/q})$
- If $q$ small, recover $a$ mod $q$
- From $a$ mod $q_i$ for every $q_i|N$, recover $a$ using CRT
- If $p-1$ is smooth (all prime factors are small) then DLP is easy!

## Random primes are not safe primes

```
p = random_prime(2**1024)
g = 2
a = random_integer(2**1024)
h = modexp(g,a,p)
```

- For random $p$, factors of $p-1$ likely to contain very small primes, some medium size prime(s) and one large prime
- Some information about $a$ can be recovered
- Safe primes: primes $p$ such that $(p-1)/2$ prime

## Small subgroup attacks

- Common recommendation (for example in DSA)
  - Choose $p$ prime such that $p-1$ has a prime factor $q$ with 160 bits
  - Choose $g$ of order $q$
  - Rationale (?): smaller exponent hence faster
- Assume that
  - Bob always uses same $p$ and same 160-bit random $b$
  - Alice chooses $g$; Bob does not check its order and Bob sends back $g^b \bmod p$
  - There exists $N|p-1$, $N \neq q$ with 160 bits and smooth (all prime factors of $N$ are small)

## Small subgroup attacks (2)

- Alice can recover $b$ as follows:
  - Send $g_i$ with small order $p_i|N$
  - Get $g_i^b$ mod $p$ from Bob
  - Solve DLP to get $b$ mod $p_i$
  - Repeat with enough small primes
  - Use Chinese remainder theorem to deduce $b$
- Countermeasure: check the order of $g$

## Outline

# Weak randomness: RSA

- Suppose Alice uses RSA private key $(p, q_a)$ and Bob uses RSA private key $(p, q_b)$. Is it safe?
- Everybody sees $n_a := pq_a$ and $n_b := pq_b$
- Alice can compute $q_b = n_b/p$
- Bob can compute $q_a = n_a/p$
- **Anyone** can compute $\gcd(n_a, n_b) = p$ and then $q_a$ and $q_b$
- Attack demonstrated in practice

    Lenstra et al. *Ron was wrong, Whit is right*
    Show that 2/1000 RSA keys are insecure

## RSA with small decryption key

- Using a small decryption for RSA is appealing for efficiency reasons, moreover
  - If $d$ has 80 bits then exhaustive search not possible
  - If $n = pq$ is large enough then factoring is not possible
- However, we then have

$$de = k\varphi(N) + 1 = k(N - z) + 1$$

  where $z = O(\sqrt{N})$ and $d, k$ are small
- "Small root problem" solved using lattice reduction, following Coppersmith's methods

## ECDSA

- Public key signature standard
- Parameters defined by
  - $H$ a hash function
  - $K$ a finite field
  - $q$ a prime
  - $E$ an elliptic curve over $K$ with $qh$ points, $h \leq 4$
  - $P$ a point of order $q$ on $E$
- Key generation
  - Choose secret key $x$ randomly in $\{1, \ldots, q-1\}$
  - Set public key $Q = xP$

## ECDSA: signature

- Let $f : E \to \mathbb{Z}_q : P = (x, y) \to x \bmod q$ where $x$ is some well-defined integer representation of the $x$-coordinate of $P$
- To sign a message $m$:
  1. Choose $k$ randomly in $\{1, \ldots, q-1\}$
  2. Let $T = kP$
  3. Let $r = f(T)$. If $r = 0$ start again
  4. Let $e = H(m)$
  5. Let $s = (e + xr)/k \bmod q$. If $s = 0$ start again
  6. Return $(r, s)$

## ECDSA: verification

- To verify signature $(r, s)$ on a message $m$
  - Reject if $r, s \notin \{1, \ldots, q-1\}$
  - Let $e = H(m)$
  - Let $u_1 = e/s \bmod q$ and $u_2 = r/s \bmod q$
  - Let $T = u_1 P + u_2 Q$
  - Accept iff $r = f(T)$

- Correctness: $u_1 + x u_2 = (e + rx)/s = k \bmod q$ hence $u_1 P + u_2 Q = T$

## ECDSA: security

- Existential unforgeability, if we replace the hash function by a random function and the group by a generic group, and suppose $f$ is "almost invertible"
- Essential that $k$ does not repeat as otherwise two signatures $(r, s)$ and $(r', s')$ give

$$x = \frac{se' - s'e}{r(s' - s)} \bmod q$$

(used to recover the secret key of Sony PS3)

- More attacks if some bits of $k$ leak or repeat (see *Lattice attacks on digital signature schemes*, by Howgrave-Graham and Smart)
- Android RNG weaknesses led to 56 bitcoin theft in 2013

## Dual EC-PRNG

- Pseudo-random number generator standardized by NIST, with a security proof
- Was known to be flawed before standardization: parameters can be constructed with a backdoor
- Other variants in the same standard were more efficient and had neither the security proof nor the backdoor issue
- Evidence that NSA acted to get it part of the standard, that they know a backdoor, and that they have used it

## Outline

## Side-channel attacks

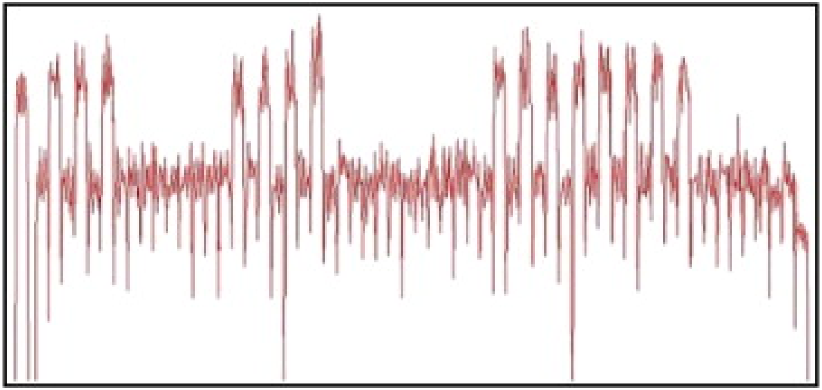
- So far we have assumed the attacker had access to some public data and some oracles, and was trying to deduce private data using mathematical algorithms
- In practice, the secret data may be on a smart card, and the attacker may observe the smart card when the computation is being done
- Does this help?

## Remember: Square-and-Multiply

- Used in RSA decryption to compute $c^d$ mod $n$, in ElGamal decryption to compute $c_2^x$ mod $p$, etc

```
1: Let $x = \sum_{i=0}^{n} x_i 2^i$
2: $g' \leftarrow g$; $c \leftarrow g^{x_0}$;
3: **for** i=1 **to** n **do**
4:     $g' \leftarrow g'^2$ mod $p$
5:     **if** $x_i = 1$ **then**
6:         $h \leftarrow hg'$ mod $p$
7:     **end if**
8: **end for**
9: **return** $h$
```

## Power consumption



- What is happening here?

## Issue and Countermeasure

- In square-and-multiply algorithm, sequence of operations performed depends on the secret
- One can distinguish operations by looking at power trace (in practice, need several measurements to reduce noise)
- Countermeasures: add dummy operations or randomness
  - Always multiply and only update register when bit is 1
  - Replace $x$ by $x + x'(p-1)$

## Hamming weight/distance models

- For AES, sequence of operations independent of secret but intermediary register values depend on secret
- Attacker needs a finer side-channel model, depending on the computing architecture
  - Hamming weight model: power consumption correlated to number of bits at 1 in a register
  - Hamming distance model: power consumption correlated to number of bits flipped in a register
  - Asymmetric model: flipping from 1 to 0 leaks more power than flipping from 0 to 1

## Side-channel attacks

- Example of successfully exploited side-channels (at least in academic contexts): power consumption, time, electromagnetic radiations, . . .
- Examples for both symmetric and public key primitives
- Do not require to break the maths, but do require some physical access to the computing device
- See Standaert, *Introduction to Side-Channel Attacks*

## Fault attacks

- Side-channel attacks are *passive* attacks: only observe the computing device
- Fault attacks modify the computation result
- Physically achieved by changing the ground power level, flashing with a laser light, changing the clock speed, etc
- How can a faulty computation be useful?
  - Work in another group of smooth order
  - Remove some access control condition
  - . . .

# Fault attack against RSA

- Suppose RSA-CRT decryption is used
  1. Compute $m_p = c^d \bmod p$ and $m_q = c^d \bmod q$
  2. Compute $m \bmod n$ with extended Euclide algorithm
- Fault attack:
  - Choose $m$
  - Send $c = m^e \bmod n$ for the device to decrypt/sign
  - Induce a fault s.t. $m_p$ replaced by $m_{p'} = c^d \bmod p'$
  - Deduce $q = \gcd(m - m', n)$

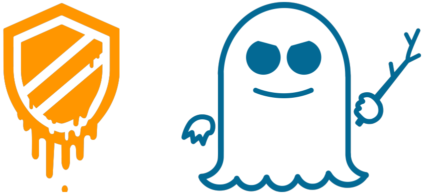
## Fault attacks: in practice

- Fault injection techniques:
  - Variations in supply voltage, clock frequency, temperature
  - Use of white light, X-ray or ion beams
- Realistic(ish) effects:
  - Set a register to 0
  - Set a register to random value
  - Set a register to arbitrary value

## Cold boot attacks

- Turn power off while computer is using key material
- Read the memory: part of key material will still be there
- Memory erasure rate is slower if cooled down

## Meltdown and Spectre



- Hardware provides isolation mechanisms between processes, used to protect cryptographic keys
- Meltdown and Spectre: can get around protection
- See `meltdownattack.com`

## The threat of quantum computers

- Quantum computers solve discrete logarithms & factoring
- Hence they break SSH, TLS , Signal, PGP, ...
- Not known: security of lattices problems, polynomial systems solving, word problem, etc
- Not known: hardness of NP-hard problems
- Not known: can (large) quantum computers be built?

## Outline

"Provable security": reductionist approach and hard problems

Example: ElGamal Encryption protocol

Limitations to the provable security approach

## Conclusion

- Security proofs in cryptography have considerable value
  - Rule out some structural weaknesses
  - Allow a more focused cryptanalysis
  - Inspire protocol design
- Yet, a “provably secure” cryptographic protocol may be very insecure
  - If the proof is wrong
  - If the “hard problem” is not hard enough
  - If the security model fails to capture the reality
- Security is a complex matter; should we say “security arguments” instead of “security proofs”?