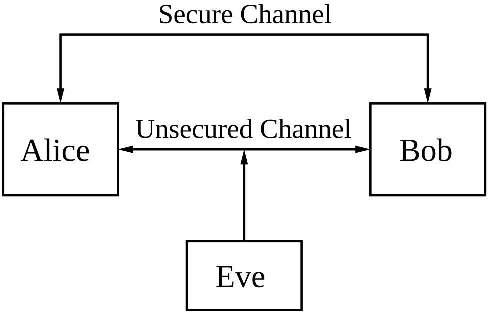
Part VI

Public-key cryptography

Drawbacks with symmetric-key cryptography

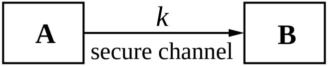
Symmetric-key cryptography: Communicating parties a priori share some secret information.



Key establishment problem

How do Alice and Bob establish the secret key k?

Method 1: Point-to-point key distribution.
(Alice selects the key and sends it to Bob over a secure channel)



The secure channel could be:

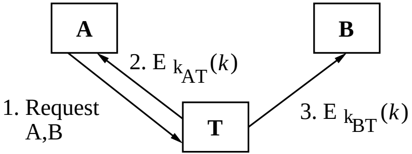
- A trusted courier.
- A face-to-face meeting in a dark alley, etc.

This is generally not practical for large-scale applications.

Key establishment problem

Method 2: Use a Trusted Third Party T.

- Each user A shares a secret key k_{AT} with T for a symmetric-key encryption scheme E.
- To establish this key, A must visit T once.
- T serves as a key distribution centre (KDC):



1. A sends T a request for a key to share with B.
2. T selects a session key k, and encrypts it for A using k_{AT}.
3. T encrypts k for B using k_{BT}.

Key establishment problem

Drawbacks of using a KDC:

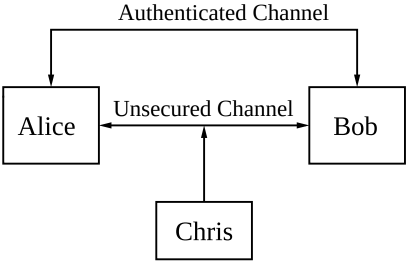
- The TTP must be unconditionally trusted.
  - Makes it an attractive target.
- Requirement for an on-line TTP.
  - Potential bottleneck.
  - Critical reliability point.

Non-Repudiation is Impractical

- Non-repudiation: Preventing an entity from denying previous actions or commitments.
  - Denying being the source of a message.
- Symmetric-key techniques can be used to achieve non-repudiation, but typically requires the services of an on-line TTP (e.g., use a message authentication code where each user shares a secret key with the TTP).

Public-key cryptography

- Public-key cryptography: Communicating parties a priori share some authenticated (but non-secret) information.



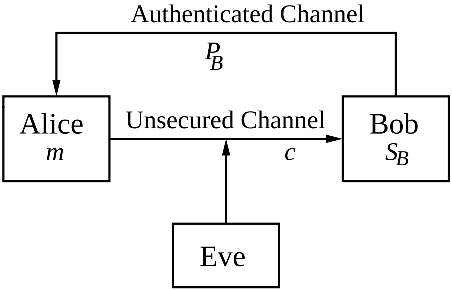
- Invented by Ralph Merkle, Whitfield Diffie, Martin Hellman in 1975.
  (And in 1970 by researchers at GCHQ.....)

Key pair generation for public-key crypto

- Each entity A does the following:
  1. Generate a key pair (P_A, S_A).
  2. S_A is A's secret key.
  3. P_A is A's public key.

- Security requirement: It should be infeasible for an adversary to recover S_A from P_A.

Public-key encryption



- To encrypt a secret message m for Bob, Alice does:
  1. Obtain an authentic copy of Bob's public key P_B.
  2. Compute $c = E(P_B, m)$; E is the encryption function.
  3. Send c to Bob.
- To decrypt c, Bob does:
  1. Compute $m = D(S_B, c)$; D is the decryption function.

Public-key vs. symmetric-key

Advantages of public-key cryptography:

- No requirement for a secret channel.
- Each user has only 1 key pair, which simplifies key management.
- Facilitates the provision of non-repudiation services (with digital signatures).

Disadvantages of public-key cryptography:

- Public keys are typically larger than symmetric keys.
- Public-key schemes are slower than their symmetric-key counterparts.

Definition of public-key cryptography

Definition: A *public-key cryptosystem* consists of:

- M – the plaintext space,
- C – the ciphertext space,
- K_{pubkey} – the space of public keys,
- K_{privkey} – the space of private keys,
- A randomized algorithm $\mathcal{G}\colon \{\mathbb{1}^{\ell} : \ell \in \mathbb{N}\} \to K_{\text{pubkey}} \times K_{\text{privkey}}$, called a *key-generation function*,
- An *encryption* algorithm $\mathcal{E}\colon K_{\text{pubkey}} \times M \to C$,
- A *decryption* algorithm $\mathcal{D}\colon K_{\text{privkey}} \times C \to M$.

Correctness requirement: For a given key pair $(k_{\text{pubkey}}, k_{\text{privkey}})$ produced by $\mathcal{G}$,

$$\mathcal{D}(k_{\text{privkey}}, \mathcal{E}(k_{\text{pubkey}}, m)) = m$$

for all $m \in M$.

The RSA encryption scheme

- Ron Rivest, Adi Shamir, and Leonard Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM **21** (2): pp. 120–126, 1978.
- Also invented by Clifford Cocks in 1973 (GCHQ).
- Key generation:
  - Choose random primes p and q with $\log_2 p \approx \log_2 q \approx 2^{\ell/2}$.
  - Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
  - Choose an integer e with $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
  - Compute $d = e^{-1} \bmod \phi(n)$. The public key is (n, e) and the private key is (n, d).
- Message space: $M = C = \mathbb{Z}_n^* = \{m \in \mathbb{Z} : 0 \leq m < n \text{ and } \gcd(m, n) = 1\}$.
- Encryption: $\mathcal{E}((n, e), m) = m^e \bmod n$.
- Decryption: $\mathcal{D}((n, d), c) = c^d \bmod n$.

Modular exponentiation

To calculate $m^e \bmod n$, use the square and multiply algorithm.

Example

- Let $n = 851$, $e = 631$, $m = 2$. Write $e = 631$ in binary:

$$631 = 2^9 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0.$$

- Compute successive powers of $m = 2$ modulo n:

$$\begin{array}{cc}
2 \equiv 2 \pmod{851} & 2^2 \equiv 4 \pmod{851} \\
2^{2^2} \equiv 16 \pmod{851} & 2^{2^3} \equiv 256 \pmod{851} \\
2^{2^4} \equiv 9 \pmod{851} & 2^{2^5} \equiv 81 \pmod{851} \\
2^{2^6} \equiv 604 \pmod{851} & 2^{2^7} \equiv 588 \pmod{851} \\
2^{2^8} \equiv 238 \pmod{851} & 2^{2^9} \equiv 478 \pmod{851}.
\end{array}$$

- Multiply:

$$\begin{aligned}
2^{631} &= 2^{2^9} \cdot 2^{2^6} \cdot 2^{2^5} \cdot 2^{2^4} \cdot 2^{2^2} \cdot 2^{2^1} \cdot 2^{2^0} \\
&\equiv 478 \cdot 604 \cdot 81 \cdot 9 \cdot 16 \cdot 4 \cdot 2 \equiv 775 \pmod{851}.
\end{aligned}$$

A framework for security definitions

Recall that for a symmetric-key encryption scheme, security depends on three questions:

1. How does the adversary interact with the communicating parties?
2. What are the computational powers of the adversary?
3. What is the adversary's goal?

- Basic assumption (Kerckhoffs's principle, Shannon's maxim): The adversary knows everything about the algorithm, except the secret key k. (Avoid security by obscurity!!)

The same principles also apply to public-key cryptography.

1. Adversary's Interaction

Possible methods of attacks against a public-key cryptosystem:

- Passive attacks:
  - Key-only attack: The adversary knows the public key(s). Equivalent to a chosen-plaintext attack, since we always assume the adversary knows the public key(s).
  - Ciphertext-only attack: The adversary knows a public key and some ciphertext(s) encrypted under the public key.
- Active attacks:
  - Chosen-ciphertext attack: The adversary can choose some ciphertext(s) and obtain the corresponding plaintext(s).
  - Adaptive chosen-ciphertext attack: Same as above, except the adversary can also choose which ciphertexts to query, based on the results of previous queries.

3. Adversary's goal

Possible goals when attacking a public-key cryptosystem:

- Total break: Determine the private key, or determine information equivalent to the private key.
- Decrypt a given ciphertext: Adversary is given a ciphertext c and decrypts it (without querying for the decryption of c).
- Decrypt a chosen ciphertext: Adversary chooses a ciphertext c and decrypts it (without querying for the decryption of c).
- Learn some partial information about a message: Adversary is given/chooses a ciphertext c and learns some partial information about the decryption of c (without querying for the decryption of c).

Chosen ciphertext security

Definition

A public-key cryptosystem is said to be secure if it is semantically secure against an adaptive chosen-ciphertext attack by a computationally bounded adversary.

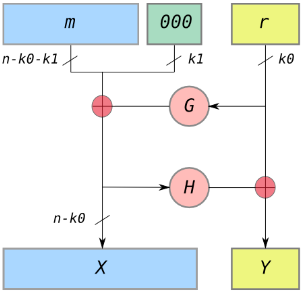
- Adaptive chosen-ciphertext attack: The adversary can choose which ciphertexts to query, based on the results of previous queries.

(Im)possibility of semantic security

A deterministic encryption algorithm (such as RSA) cannot yield semantic security.

- Given a ciphertext c and a public key, choose m at random and compute $c' = E_{\text{pubkey}}(m)$.
- If $c = c'$ then we know the plaintext was m.
- If $c \neq c'$ then we know the plaintext was **not** m.
- Either way, we have learned information about the plaintext.

Optimal Asymmetric Encryption Padding



- Public key (n, e)
- Private key (n, d)
- $k, k_0, k_1 \in \mathbb{N}$ with $k + k_0 + k_1 = \log_2 n$
- Hash function
  $G\colon \{0,1\}^{k_0} \to \{0,1\}^{k+k_1}$
- Hash function
  $H\colon \{0,1\}^{k+k_1} \to \{0,1\}^{k_0}$
- Encryption: To encrypt $m \in \{0,1\}^k$:
  - $s \leftarrow (m \parallel 0^{k_1}) \oplus G(r)$
  - $\mathcal{E}(m) \leftarrow (s \parallel H(s) \oplus r)^e \bmod n.$
- Decryption:
  - $s \parallel t \leftarrow c^d \bmod n$
  - $m \parallel 0^{k_1} \leftarrow H(s) \oplus t$
  - Check that $H(s) \oplus t$ ends with 0^{k_1}!
  - If so, output $\mathcal{D}(c) = m$; otherwise, return error.

Part VII

Digital signatures

Definition of public-key cryptography

Recall that a *public-key cryptosystem* consists of:

- M – the plaintext space,
- C – the ciphertext space,
- K_{pubkey} – the space of public keys,
- K_{privkey} – the space of private keys,
- A randomized algorithm $\mathcal{G}\colon \{\mathbb{1}^\ell : \ell \in \mathbb{N}\} \to K_{\text{pubkey}} \times K_{\text{privkey}}$, called a *key-generation function*,
- An *encryption* algorithm $\mathcal{E}\colon K_{\text{pubkey}} \times M \to C$,
- A *decryption* algorithm $\mathcal{D}\colon K_{\text{privkey}} \times C \to M$.

Correctness requirement: For a given key pair $(k_{\text{pubkey}}, k_{\text{privkey}})$ produced by $\mathcal{G}$,

$$\mathcal{D}(k_{\text{privkey}}, \mathcal{E}(k_{\text{pubkey}}, m)) = m$$

for all $m \in M$.

Motivation for digital signatures

- In the definition of a public-key cryptosystem, decryption must be a left inverse of encryption:

$$\mathcal{D}(k_{\text{privkey}}, \mathcal{E}(k_{\text{pubkey}}, m)) = m.$$

- There is no corresponding requirement that decryption be a **right** inverse of encryption:

$$\mathcal{E}(k_{\text{pubkey}}, \mathcal{D}(k_{\text{privkey}}, c)) \stackrel{?}{=} c.$$

- In some cases (e.g. plain RSA without padding), decryption is in fact a right inverse of encryption.
- In other cases (e.g. ElGamal), decryption is not a right inverse of encryption.
- When decryption is a right inverse of encryption, we get a useful construction: digital signatures

RSA Signature Scheme

Ron Rivest, Adi Shamir, and Leonard Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM **21** (2): pp. 120–126, 1978.

Key generation: Same as in RSA encryption.

Signature generation: To sign a message m:

1. Compute $s = m^d \bmod n$.
2. The signature on m is s.

Signature verification: To verify a signature s on a message m:

1. Obtain an authentic copy of the public key (n, e).
2. Compute $s^e \bmod n$
3. Accept (m, s) if and only if $s^e \bmod n = m$.

Definition of digital signatures

Definition: A *digital signature scheme* consists of:

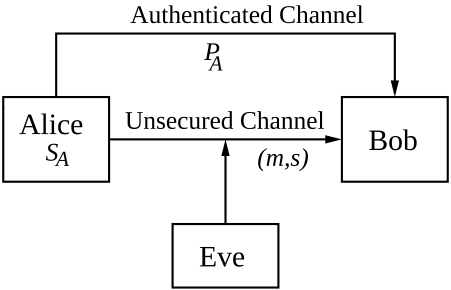
- M – the plaintext space,
- S – the signature space,
- K_{pubkey} – the space of public keys,
- K_{privkey} – the space of private keys,
- A randomized algorithm $\mathcal{G}\colon \{\mathbb{1}^\ell : \ell \in \mathbb{N}\} \to K_{\text{pubkey}} \times K_{\text{privkey}}$, called a *key-generation function*,
- A *signing* algorithm $\mathcal{S}\colon K_{\text{privkey}} \times M \to S$,
- A *verification* algorithm $\mathcal{V}\colon K_{\text{pubkey}} \times M \times S \to \{\textbf{true}, \textbf{false}\}$.
- A *valid* signature is one which verifies. An *invalid* signature is one which does not verify.

Correctness requirement: For a given key pair $(k_{\text{pubkey}}, k_{\text{privkey}})$ produced by $\mathcal{G}$,

$$\mathcal{V}(k_{\text{pubkey}}, m, \mathcal{S}(k_{\text{privkey}}, m)) = \textbf{true}$$

for all $m \in M$.

Digital signatures



- To sign a message m, Alice does:
  1. Compute $s = \text{Sign}(S_A, m)$.
  2. Send m and s to Bob.
- To verify Alice's signature s on m, Bob does:
  1. Obtain an authentic copy of Alice's public key P_A.
  2. Accept if $\text{Verify}(P_A, m, s) = \text{Accept}$.

Basic security requirements

Goals of a digital signature scheme:

- *Authenticate* the origin of a message.
- Guarantee the *integrity* of a message.
- Basic security requirements:
  - It should be infeasible to deduce the private key from the public key.
  - It should be infeasible to generate valid signatures without the private key.

Goals of the Adversary

1. Total break: E recovers A's private key, or a method for systematically forging A's signatures (i.e., E can compute A's signature for arbitrary messages).
2. Selective forgery: E forges A's signature for a selected subset of messages.
3. Existential forgery: E forges A's signature for a single message; E may not have any control over the content or structure of this message.

Attack Model

Types of attacks E can launch:

1. Key-only attack: The only information E has is A's public key.
2. Known-message attack: E knows some message/signature pairs.
3. Chosen-message attack: E has access to a signing oracle which it can use to obtain A's signatures on some messages of its choosing.

Security Definition

Definition: A signature scheme is said to be secure if it is existentially unforgeable by a computationally bounded adversary who launches a chosen-message attack.
Note: The adversary has access to a signing oracle. Its goal is to compute a single valid message/signature pair for any message that was not previously given to the signing oracle.

Existential forgery against RSA

Even if the RSA problem is intractable, the basic RSA scheme is still insecure. Here is an existential forgery under a key-only attack:

- Select $s \in \mathbb{Z}_n$ with $\gcd(s, n) = 1$.
- Compute $s^e \bmod n$.
- Set $m = s^e \bmod n$.
- Then s is a valid signature for m.

Here is a selective forgery under a chosen message attack. Given $m \in \mathbb{Z}_n$ with $\gcd(m, n) = 1$:

- Compute $m' = 2^e \cdot m \bmod n$
- Request the signature s' of m'
- Compute $s = s'/2 \bmod n$.
- Then s is a valid signature for m.

Full Domain Hash RSA (RSA-FDH)

Let $H\colon \{0,1\}^* \to \mathbb{Z}_n$ be a hash function.

Key generation: Same as in RSA.

Signature generation: To sign a message $m \in \{0,1\}^*$:

1. Compute $s = H(m)^d \bmod n$.
2. The signature on m is s.

Signature verification: To verify a signature s on a message m:

1. Obtain an authentic copy of the public key (n, e).
2. Compute $s^e \bmod n$
3. Accept (m, s) if and only if $s^e \bmod n = H(m)$.

Security of RSA-FDH

Theorem (Bellare & Rogaway, 1996): If the RSA problem is intractable and H is a random function, then RSA-FDH is a secure signature scheme.

Note: This theorem does NOT always hold if H is not a random function!

Part VIII

Side-channel attacks

Cryptography as a black box

Up to this point:

- We have treated cryptography as a black box.
- We assume the attacker can observe and/or manipulate inputs and outputs.
- We do NOT assume the attacker can view or manipulate intermediate results.
- We have formal definitions of security, and we can prove security under reasonable mathematical assumptions.

Problems with the black-box viewpoint

For a cryptographer:

Formal security model + security proof = complete victory

And yet ...

- In practice, things still get broken.
- Assumptions in the security model often do not hold in reality.
- Attackers always exploit the weakest link. That weak link is almost never (black-box) crypto.
- In many systems, implementation is the weak link, and it is what gets attacked.

Overview of side channel attacks

A side-channel attack is some attack that involves observing and/or manipulating intermediate results in a cryptographic calculation.

How does one observe internal state information?

- Timing information: time how long a computation takes.
- Power consumption: monitor the amount of power used.
- Electromagnetic radiation: monitor the noise leaked by a hardware circuit.
- Acoustic information: record sound with a microphone.
- Other: cache-miss rate, in-circuit emulators, etc.

How does one manipulate internal state information?

- Fault injection
- Row hammer

Types of side-channel attacks

- Passive attacks: Attacker can manipulate inputs, and observe intermediate results.
- Active attacks: Attacker can manipulate inputs, and manipulate intermediate results.
- Fault attacks: Attacker can set input values and/or intermediate result values to invalid values.
- Physical attacks: Attacker can take apart the hardware, dunk it in an acid bath, etc.

Simple Power Analysis

Paul Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", Crypto '96.

- Many smart cards contain an RSA private key which is used to generate RSA signatures to authenticate the card.
- A counterfeiter is not supposed to be able to extract the RSA private key from the card.
- Suppose the card utilizes some measurable resource, in some data-dependent way, e.g.:
  - Amount of time it takes to perform a signature.
  - Amount of power consumed during the signature process, as a function of time.
- By measuring resource consumption, it is possible to determine the value of the private key.

Square-and-multiply algorithm

Recall the `square-and-multiply` algorithm:

Algorithm 1 Algorithm for computing $m^d \bmod n$.

```
if d = 0 then
    output 1
else if d is even then
    output (m^(d/2) mod n)^2 mod n
else if d is odd then
    output (m · (m^(d−1) mod n)) mod n
```

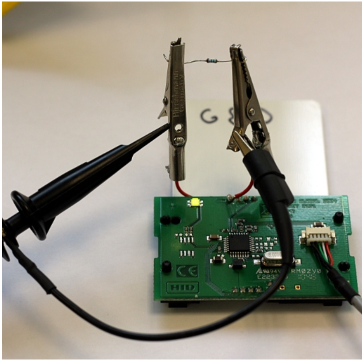
- Suppose that squaring mod n consumes different resources from (non-squaring) multiplication mod n.
- By measuring resource consumption, one can determine individual bits in d.
- A similar attack works against the `double-and-add` algorithm on elliptic curves.

Attack example

- Suppose $d = 26$ (in binary: $26 = 11010_2$).
- Then
$$m^{26} = (m\times((m\times(1\times m)^2)^2)^2)^2$$
- The computation proceeds from the inside out:
$$\mathtt{M\ S\ M\ S\ S\ M\ S} = \underbrace{\mathtt{M\ S}}_{1}\ \underbrace{\mathtt{M\ S}}_{1}\ \underbrace{\mathtt{S}}_{0}\ \underbrace{\mathtt{M}}_{1}\ \underbrace{\mathtt{S}}_{0}$$
- Similarly, in elliptic curve cryptography:
$$26 \cdot P = 2\cdot(P+2\cdot(2\cdot(P+2\cdot(0+P))))$$
and the order of operations is:
$$\mathtt{A\ D\ A\ D\ D\ A\ D} = \underbrace{\mathtt{A\ D}}_{1}\ \underbrace{\mathtt{A\ D}}_{1}\ \underbrace{\mathtt{D}}_{0}\ \underbrace{\mathtt{A}}_{1}\ \underbrace{\mathtt{D}}_{0}$$

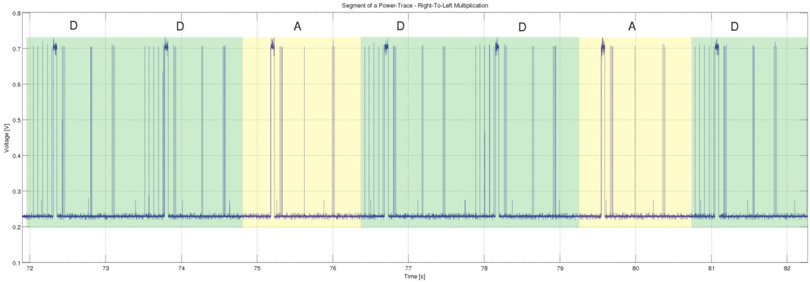
Obtaining a power trace

Credit: Alexander Petric (`http://www.alexander-petric.com/2011/08/side-channel-attack-measurement-setup-2.html`)



- Put the smart card in a card reader.
- Attach a scope to the power supply.
  - This step may involve (partially) disassembling the card reader. Note however the card itself need not be taken apart.
- Record the power consumption as a function of time.
- Make an educated guess as to which portions of the power trace correspond to which operations.

Measured power traces

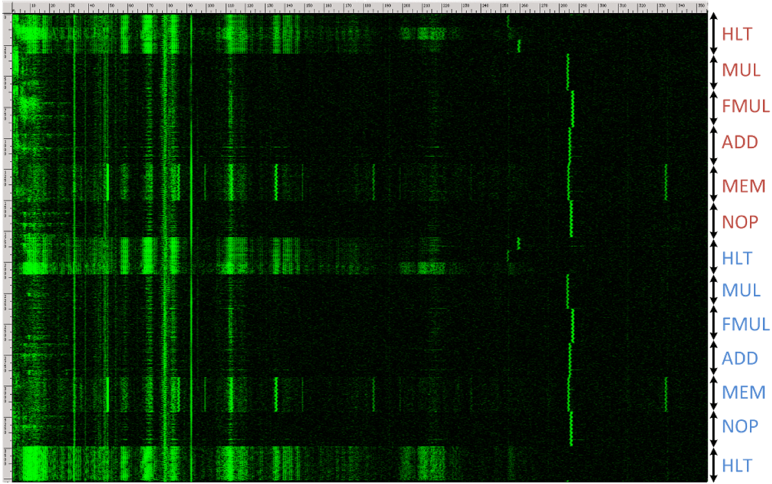


Acoustic side-channels

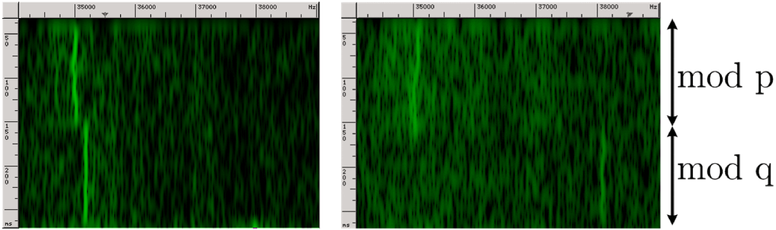
D. Genkin, A. Shamir, and E. Tromer, "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis," CRYPTO 2014.

- *Audio recordings* are a potential source of side-channel information!
- Using its built-in microphone, a mobile phone placed next to a laptop can determine a secret key used in a computation on the laptop.

Acoustic traces



Acoustic traces



(a) attacked bit is zero

(b) attacked bit is one

Cache-based side-channels

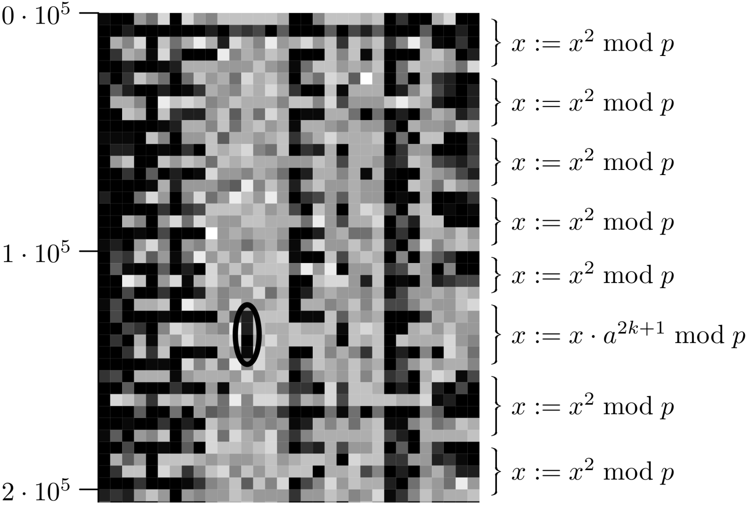
C. Percival, "Cache-missing for fun and profit,"
http://www.daemonology.net/papers/htt.pdf

- A user controlling one core of a multi-core processor can spy on processes being executed on the other core, using cache hit rate as a side channel.

If $(BIG_COMPANY) hosts their servers on Amazon:

- Buy an account on Amazon.
- Repeat (and/or wait) until your server lands on another CPU core on the same machine as $(BIG_COMPANY)'s servers.
- Steal their keys.

Cache trace


$0 \cdot 10^5$
$1 \cdot 10^5$
$2 \cdot 10^5$
$x := x^2 \bmod p$
$x := x^2 \bmod p$
$x := x^2 \bmod p$
$x := x^2 \bmod p$
$x := x^2 \bmod p$
$x := x \cdot a^{2k+1} \bmod p$
$x := x^2 \bmod p$
$x := x^2 \bmod p$

Side-channel attack countermeasures

The basic idea is to make all calculations consume constant resources independent of the input data. Some options include:

- Unified formulas: Use identical formulas for addition and doubling, or for squaring and multiplication.
- Dummy operations: Insert extra useless operations to make the calculation uniform (and discard the result).
- Double and always add: Perform the same operations independent of data values.