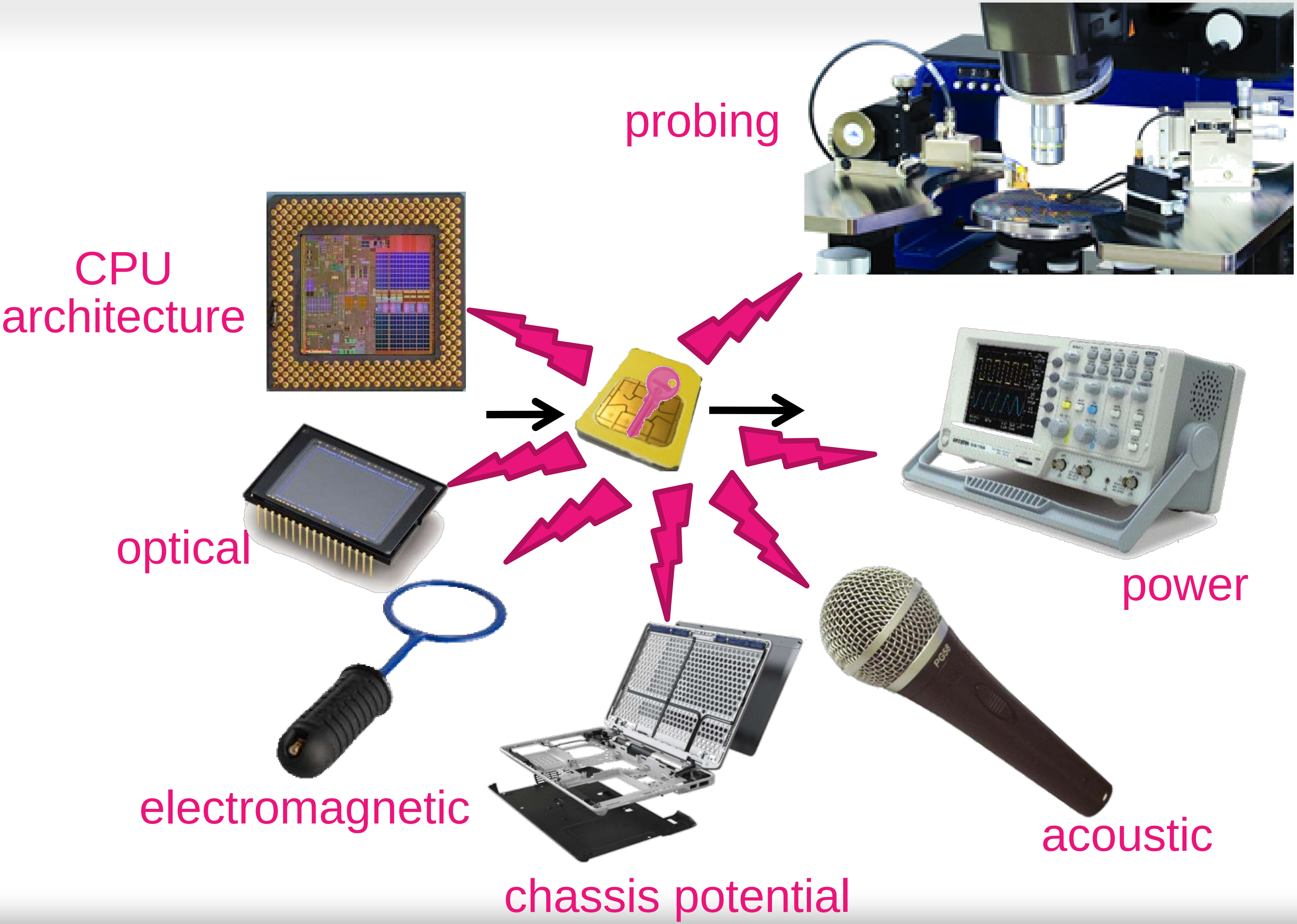
Information Security – Theory vs. Reality

0368-4474, Winter 2015-2016

Lecture 6:
Physical Side Channel Attacks on PCs

Guest lecturer:
Lev Pachmanov

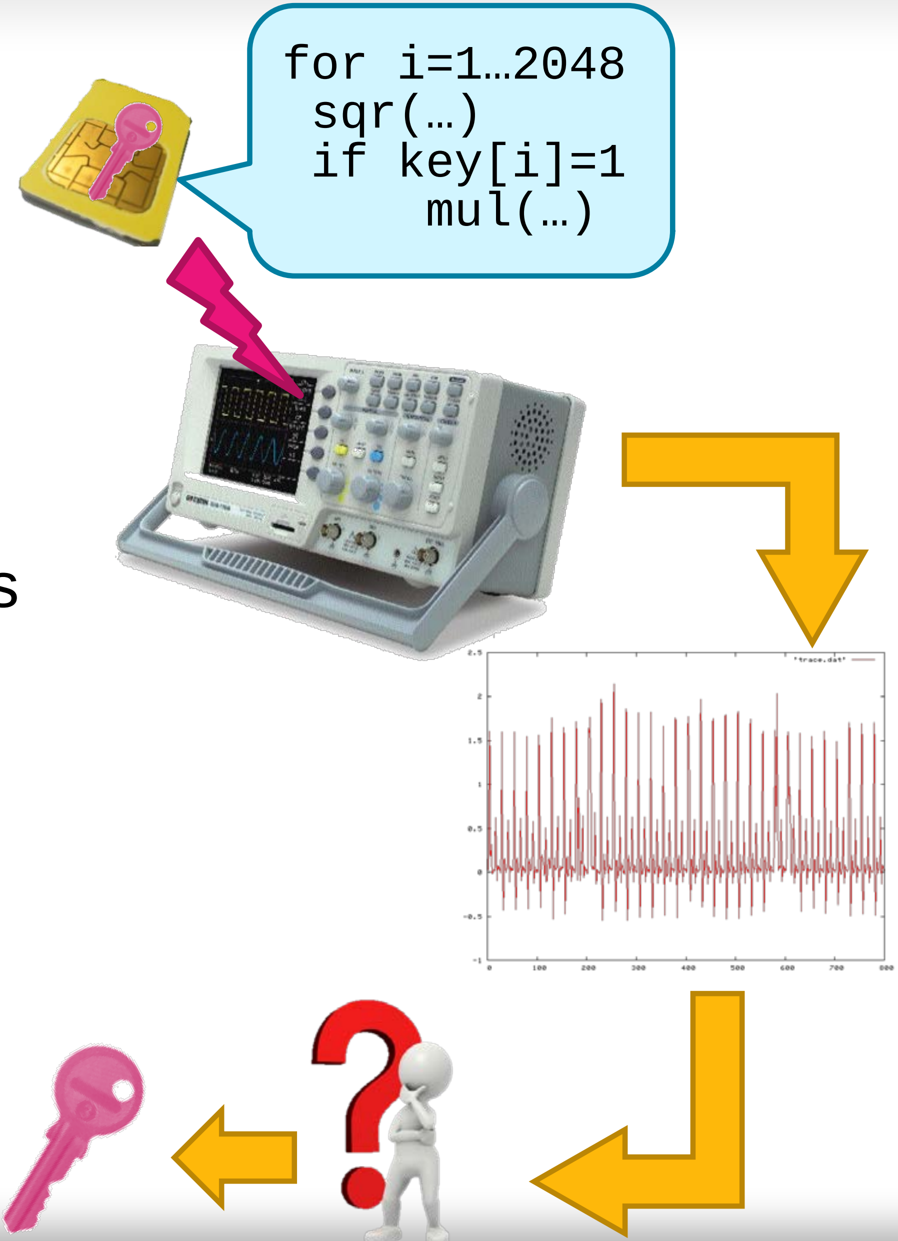
Side channel attacks



Traditional side channel attacks methodology

1. Grab/borrow/steal device
2. Find key-dependent instruction
3. Record emanations using high-bandwidth equipment (> clock rate , PC: >2GHz)
4. Obtain traces
5. Signal and cryptanalytic analysis
6. Recover key

Hard for PCs



Traditional side channel attacks methodology

1. Grab/borrow/steal device
2. Find key-dependent instruction
3. Record emanations using high-bandwidth equipment (> clock rate , PC: >2GHz)
4. Obtain traces
5. Signal and cryptanalytic analysis
6. Recover key

Hard for PCs 

Traditional side channel attacks methodology

1. Grab/borrow/steal device
2. Find key-dependent instruction
3. Record emanations using high-bandwidth equipment (> clock rate , PC: >2GHz)
4. Obtain traces
5. Signal and cryptanalytic analysis
6. Recover key

Not handed out

vs.

Measuring a 2GHz PC requires expansive and bulky equipment (compared to a 100 MHz smart card)

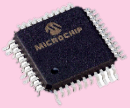
100,000$

vs.

1,000$

Complex electronics running complicated software (in parallel)

vs.



Our results

- Channels for attacking PCs
  - Ground potential (chassis and others)
  - Power
  - Electromagnetic
  - Acoustic

- Exploited via low-bandwidth cryptanalytic attacks
  - Adaptive attack (50 kHz bandwidth) [Genkin Shamir Tromer '14]
  - Non-adaptive attacks (1.5 MHz bandwidth) [Genkin Pipman Tromer '14] [Genkin Pachmanov Pipman Tromer '15]

- Common cryptographic software
  - GnuPG 1.4.13-1.4.16 (CVE 2013-4576, 2014-3591, 2014-5270)
  - RSA and ElGamal, various implementations
  - Worked with GnuPG developers to mitigate the attacks
- Applicable to various laptop models

Chassis-potential channel

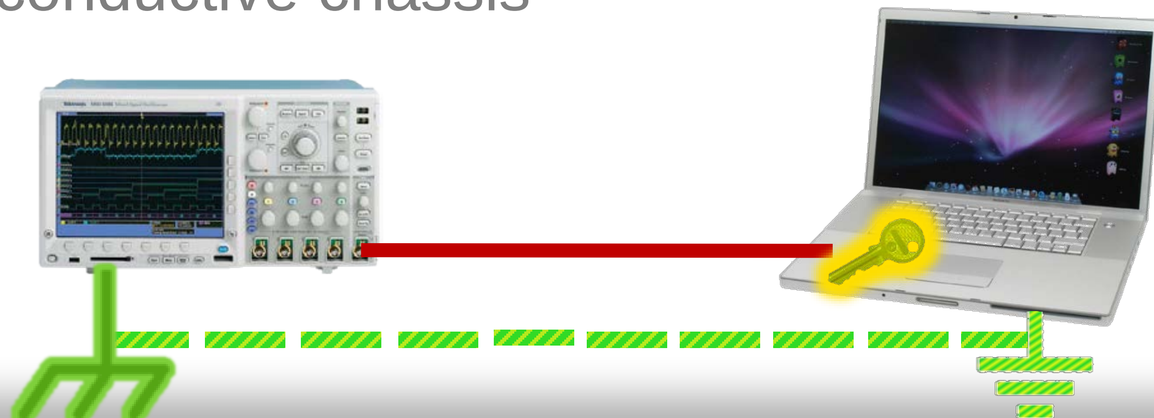
Ground-potential analysis

- **Attenuating EMI emanations**
  “Unwanted currents or electromagnetic fields? Dump them to the circuit ground!”
  (Bypass capacitors, RF shields, …)

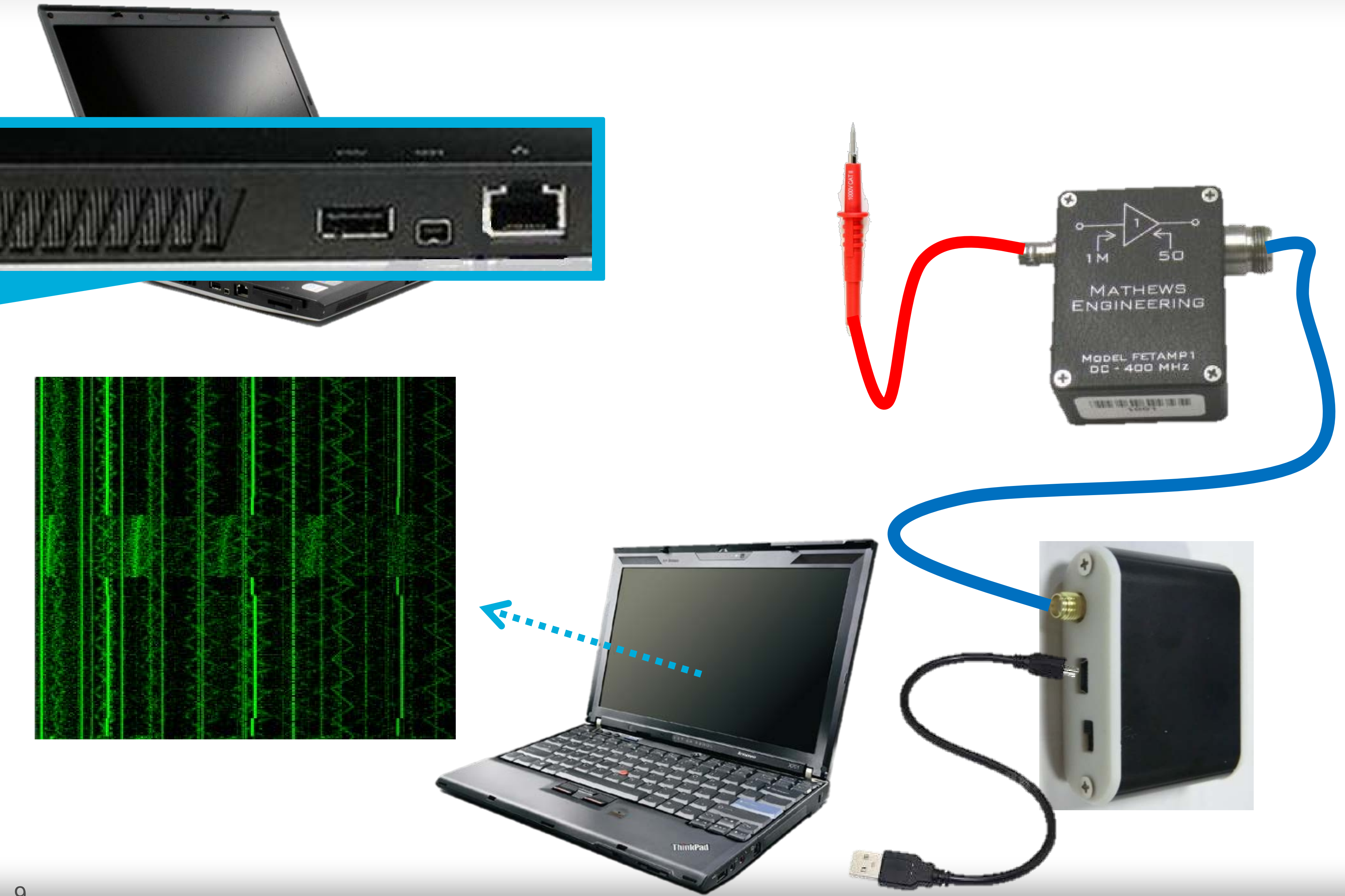
- Device is grounded, but its “ground” potential fluctuates relative to the mains earth ground.

	Computation
affects	currents and EM fields
dumped to	device ground
connected to	conductive chassis

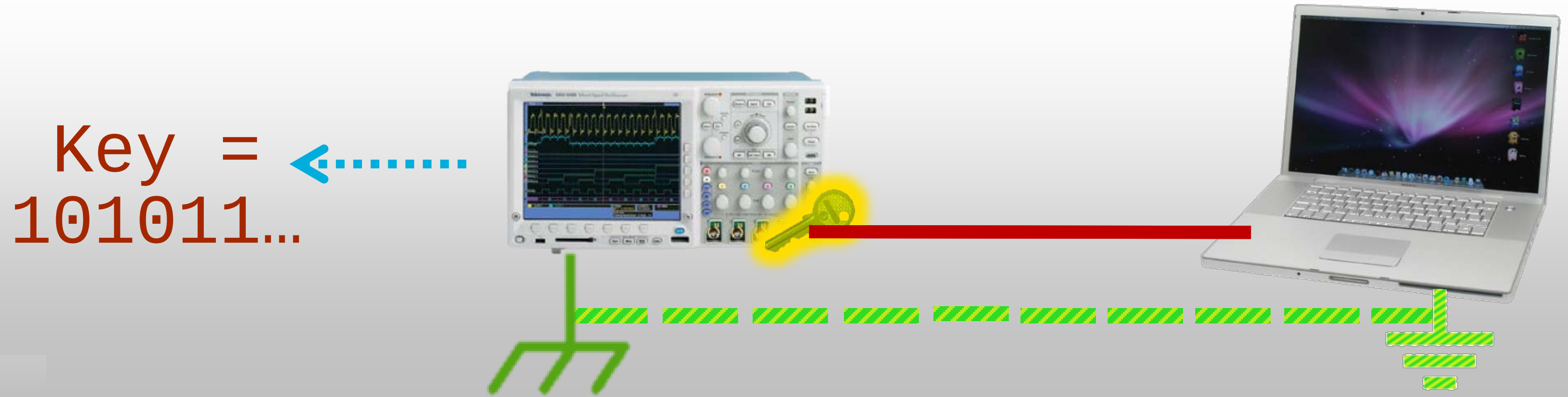
Key = 101011…



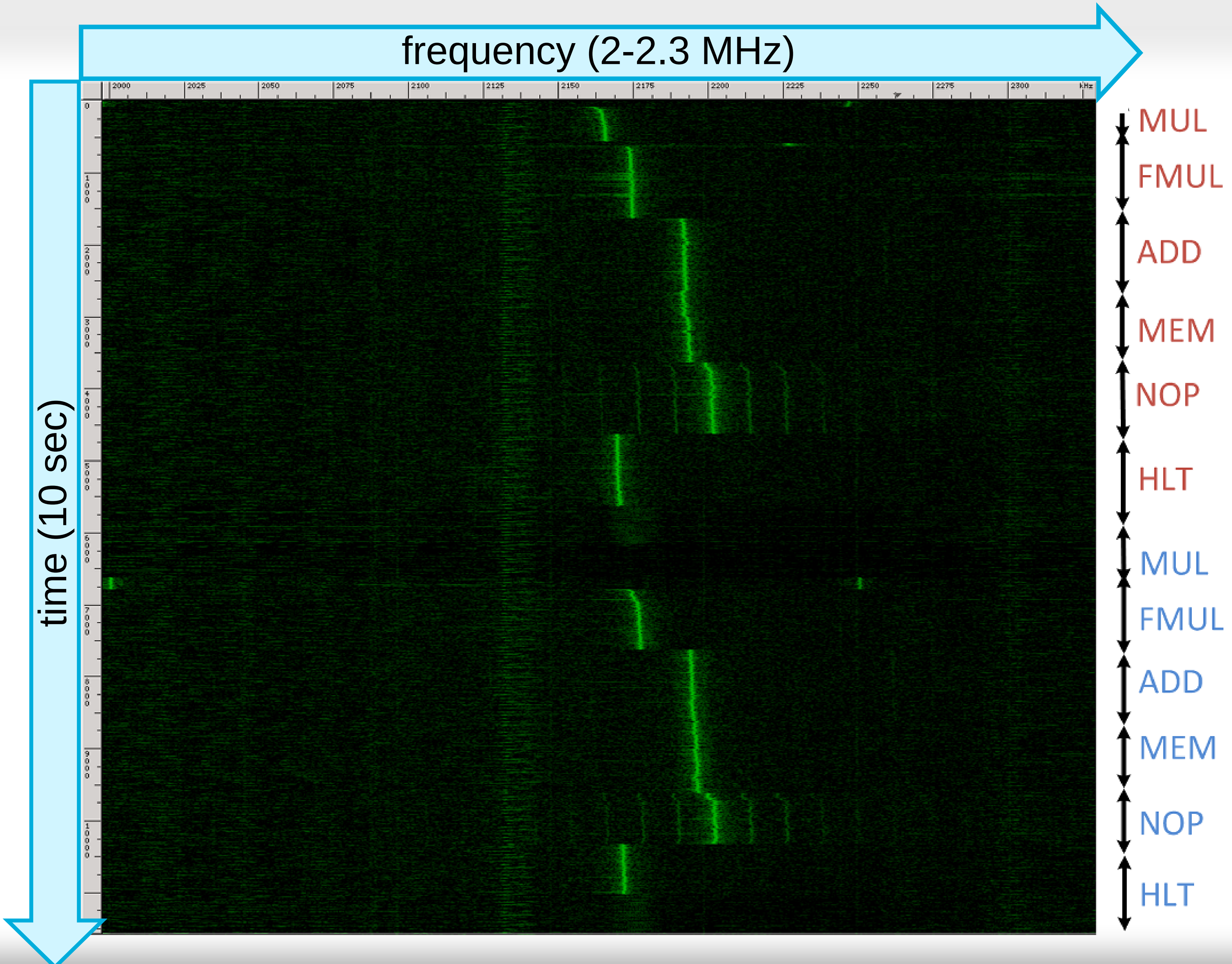
Connecting to the chassis



Demo:
distinguishing instructions



Distinguishing various CPU operations



Low-bandwidth leakage of RSA

Definitions (RSA)

Key setup

- **sk:** random primes p, q, private exponent d
- **pk:** $n = pq$, public exponent e

Encryption

$$c = m^e \bmod n$$

Decryption

$$m = c^d \bmod n$$

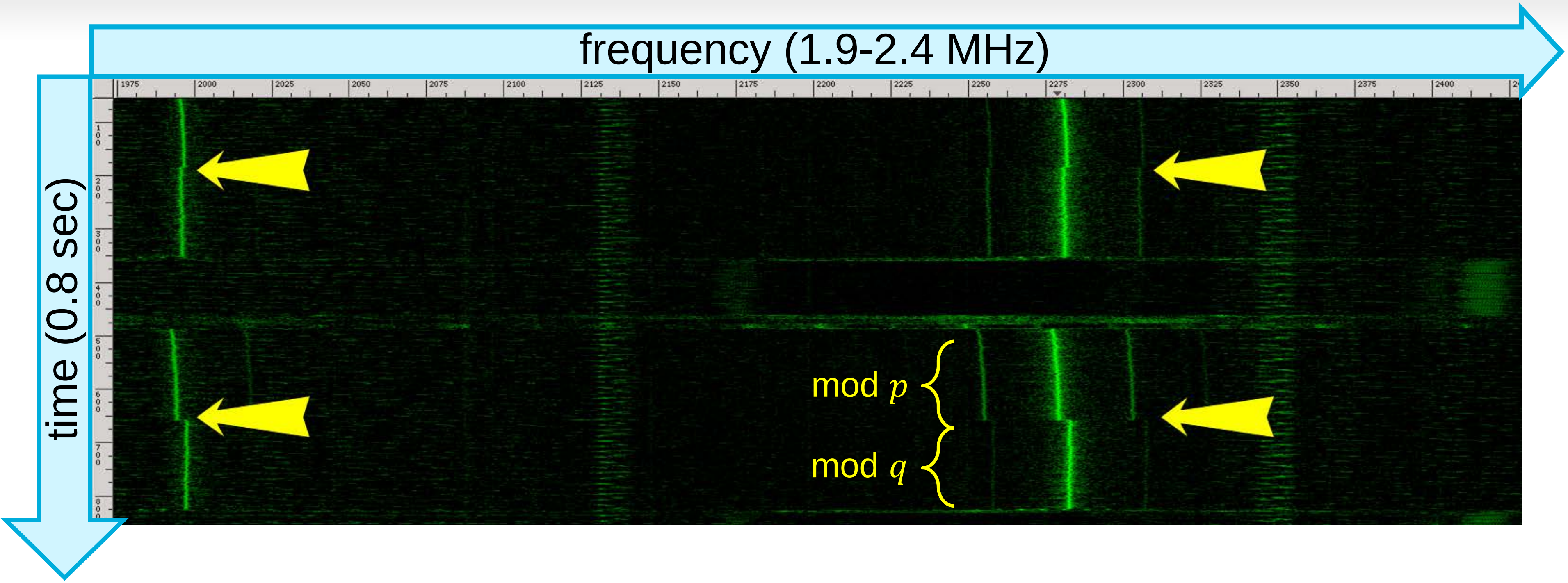
A quicker way used by most implementations

$$m_p = c^{d_p} \bmod p$$

$$m_q = c^{d_q} \bmod q$$

Obtain m using Chinese Remainder Theorem

GnuPG RSA key distinguishability



Can distinguish between:

1. Decryptions and other operations
2. Two exponentiations (mod p, mod q)
3. Different keys
4. Different primes

Key extraction

GnuPG modular exponentiation

```
modular_exponentiation(c,d,p){
 m=1
 for i=n to 1 do
  m = m² mod p
  t = m*c mod p //always mult
  if d[i]==1 then
    m=t
 return m
}
```

$m = c^{d_n \cdots d_{i+1}} \; mod \; p$

$m = c^{d_n \cdots d_{i+1} 0} \; mod \; p$

$t = c^{d_n \cdots d_{i+1} 1} \; mod \; p$

$m = c^{d_n \cdots d_i} \; mod \; p$

Q: Why always compute $t \leftarrow m \cdot c$ then conditionally copy?
A: This is a side channel countermeasure meant to protect d

no key dependent operation to measure

GnuPG modular exponentiation

```
modular_exponentiation(c,d,p){
 m=1
 for i=n to 1 do
  m = m² mod p
  t = m*c mod p //always mult
  if d[i]==1 then
    m=t
 return m
```

m depends on both $d[i]$ and c

m is squard in next iteration of the main loop

craft c to affect the squaring in the next loop iteration, based on $d[i]$

measure changes inside squaring operation and obtain $d[i]$

2GHz CPU speed vs. 1.5MHz measurements

can only see drastic changes inside squaring operation

Idea: <u>leakage self-amplification</u>

abuse algorithm's own code to amplify its own leakage!

1. Craft suitable cipher-text to affect the inner-most loop
2. Small differences in repeated inner-most loops cause a big overall difference in code behavior

Non-adaptive key extraction (similar to [Yen, Lien, Moon and Ha 05])

```
modular_exponentiation(c,d,p){
  m=1
  for i=n to 1 do
   m = m² mod p
   t = m*c mod p //always mult
   if d[i]==1 then
     m=t
  return m
}
```

$c \equiv -1 \pmod{p}$

$m \equiv 1 \pmod{p}$

$t \equiv -1 \pmod{p}$

$m \equiv \pm 1$

Many zeros or random looking, based on $d[i]$

```
karatsuba_sqr( m ){
 ...
 basic_sqr( x )
 ...
}
```

0/$

```
basic_sqr( x ){
 ...
  if( x[j]==0)
   y = 0
  else
   y = x[j]*x
}
```

If $d[i] == 1$ then $m \equiv -1 \pmod{p}$ so bits of m are “random”.

If $d[i] == 0$ then $m \equiv 1 \pmod{p}$ so bits of m have many zeros.

repeated 189 times per bit of d

~0.2ms of measurement per bit of d

A chosen ciphertext attack

Non-adaptive ciphertext choice $c \equiv -1 \bmod p$
(similar to [YLMH05]):

- RSA: $c = N - 1$
- ElGamal: $c = p - 1$

Overall attack performance:

Algorithm	Attack type	# ciphertexts	Time	BW	Cipher	Ref
Sqr-and-always-mlt	Non-adaptive chosen ciphertext	1	3 sec	2 MHz	ElGamal, RSA	[GPT14]

A chosen ciphertext attack

Non-adaptive ciphertext choice $c \equiv -1 \bmod p$ (similar to [YLMH05]):

- RSA: $c = N - 1$
- ElGamal: $c = p - 1$

Overall attack performance:

Algorithm	Attack type	# ciphertexts	Time	BW	Cipher	Ref
Sqr-and-always-mlt	Non-adaptive chosen ciphertext	1	3 sec	2 MHz	ElGamal, RSA	[GPT14]
Sliding / fixed window	Non-adaptive chosen ciphertext	2^{w-1} (usually 8 or 16)	30 sec	2 MHz	ElGamal, RSA	[GPPT15]

A chosen ciphertext attack

Non-adaptive ciphertext choice $c \equiv -1 \bmod p$ (similar to [YLMH05]):

- RSA: $c = N - 1$
- ElGamal: $c = p - 1$

Overall attack performance:

Algorithm	Attack type	# ciphertexts	Time	BW	Cipher	Ref
Sqr-and-always-mlt	Non-adaptive chosen ciphertext	1	3 sec	2 MHz	ElGamal, RSA	[GPT14]
Sliding / fixed window	Non-adaptive chosen ciphertext	2^{w-1} (usually 8 or 16)	30 sec	2 MHz	ElGamal, RSA	[GPPT15]
Sqr-and-always-mlt	Adaptive chosen ciphertext	$\frac{Key\ size}{4}$	1 hour	50 kHz	RSA	[GST14]

A chosen ciphertext attack

Non-adaptive ciphertext choice $c \equiv -1 \bmod p$ (similar to [YLMH05]):

- RSA: $c = N - 1$
- ElGamal: $c = p - 1$

Overall attack performance:

Algorithm	Attack type	# ciphertexts	Time	BW	Cipher	Ref
Sqr-and-always-mlt	Non-adaptive chosen ciphertext	1	3 sec	2 MHz	ElGamal, RSA	[GPT14]
Sliding / fixed window	Non-adaptive chosen ciphertext	2^{w-1} (usually 8 or 16)	30 sec	2 MHz	ElGamal, RSA	[GPPT15]
Sqr-and-always-mlt	Adaptive chosen ciphertext	$\frac{Key\ size}{4}$	1 hour	50 kHz	RSA	[GST14]

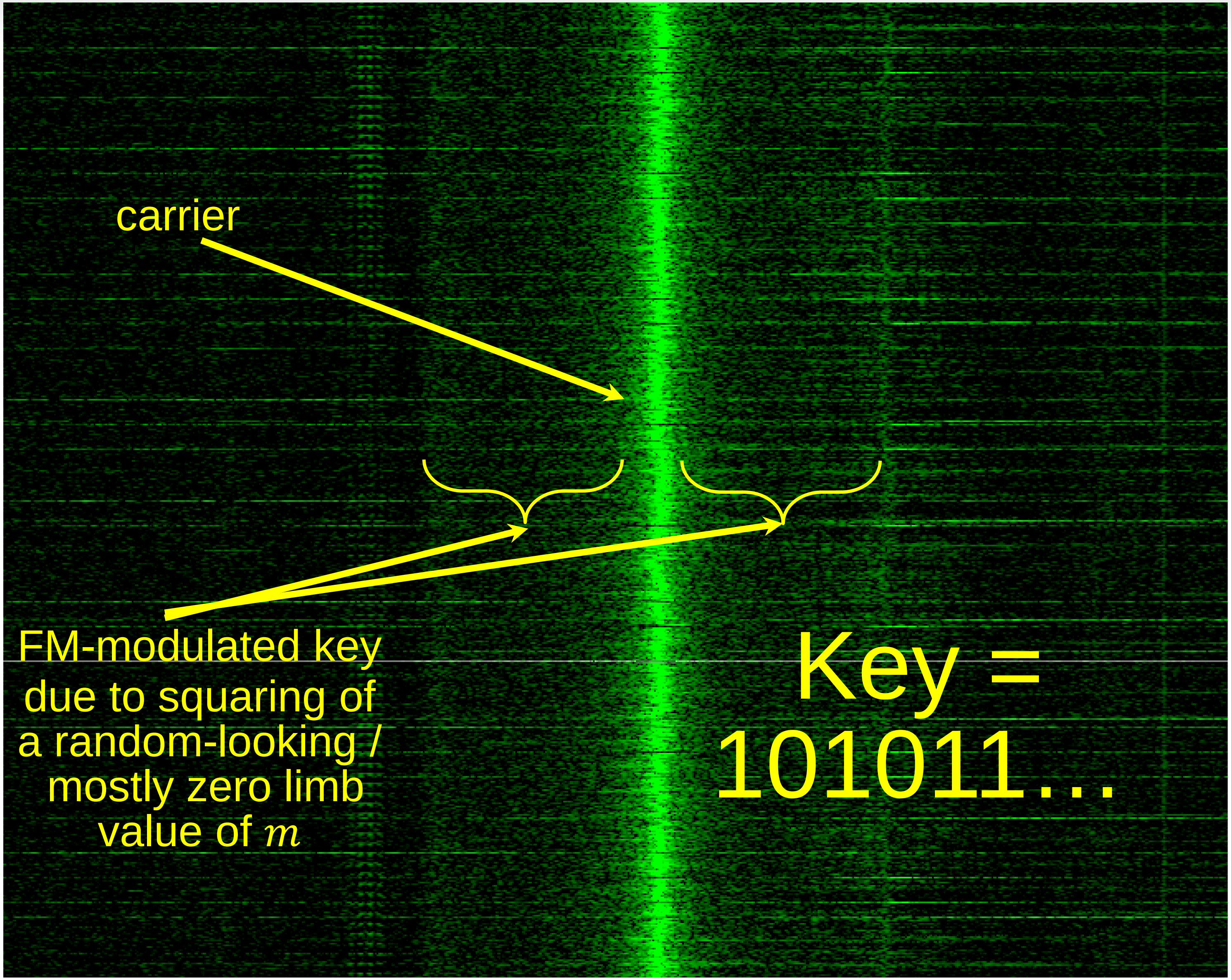
Ciphertext injection

Send chosen ciphertexts via email (PGP/MIME). Decrypted by email client (e.g., Enigmail) automatically.

Empirical results:
ground-potential attacks

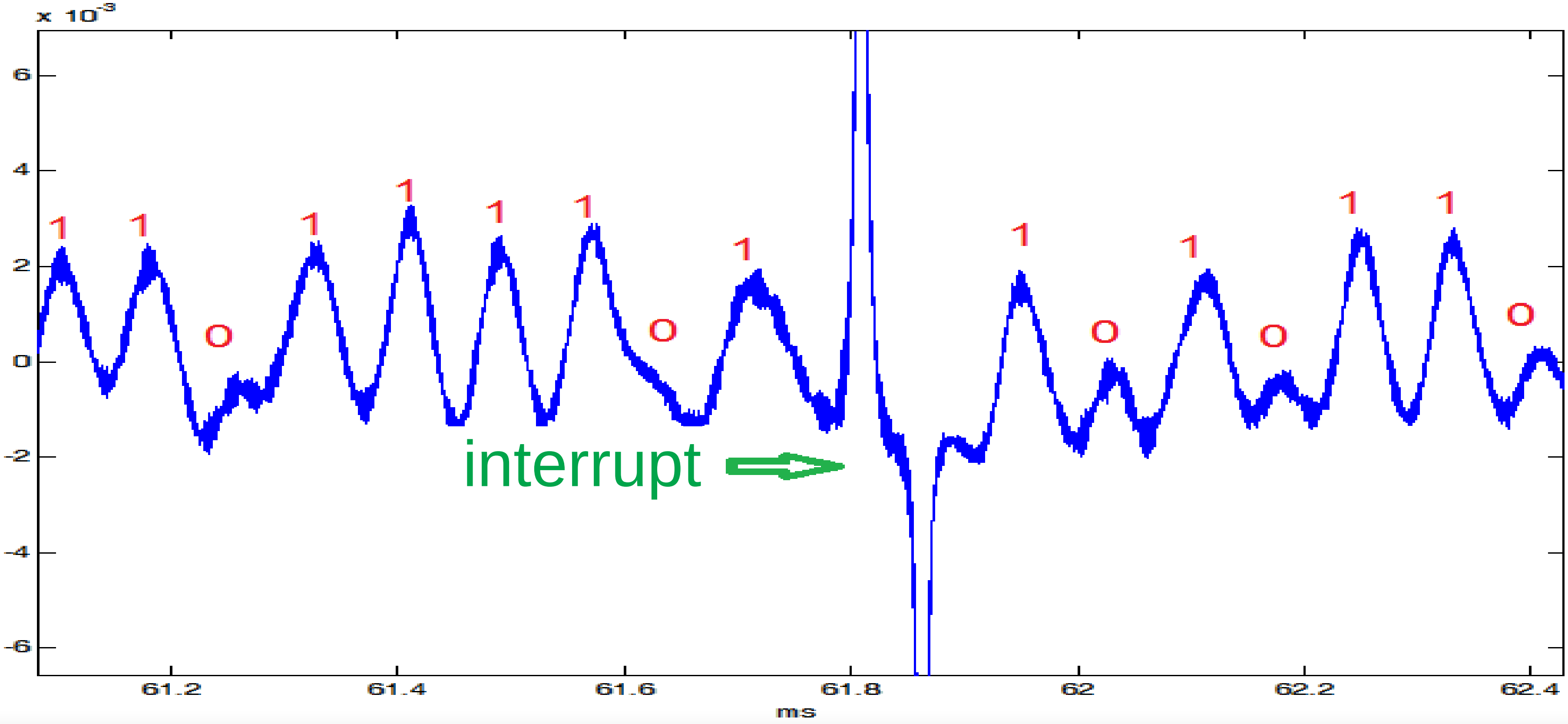
Demo:
RSA key extraction
from chassis potential

Reading the secret key (non-adaptive attack)

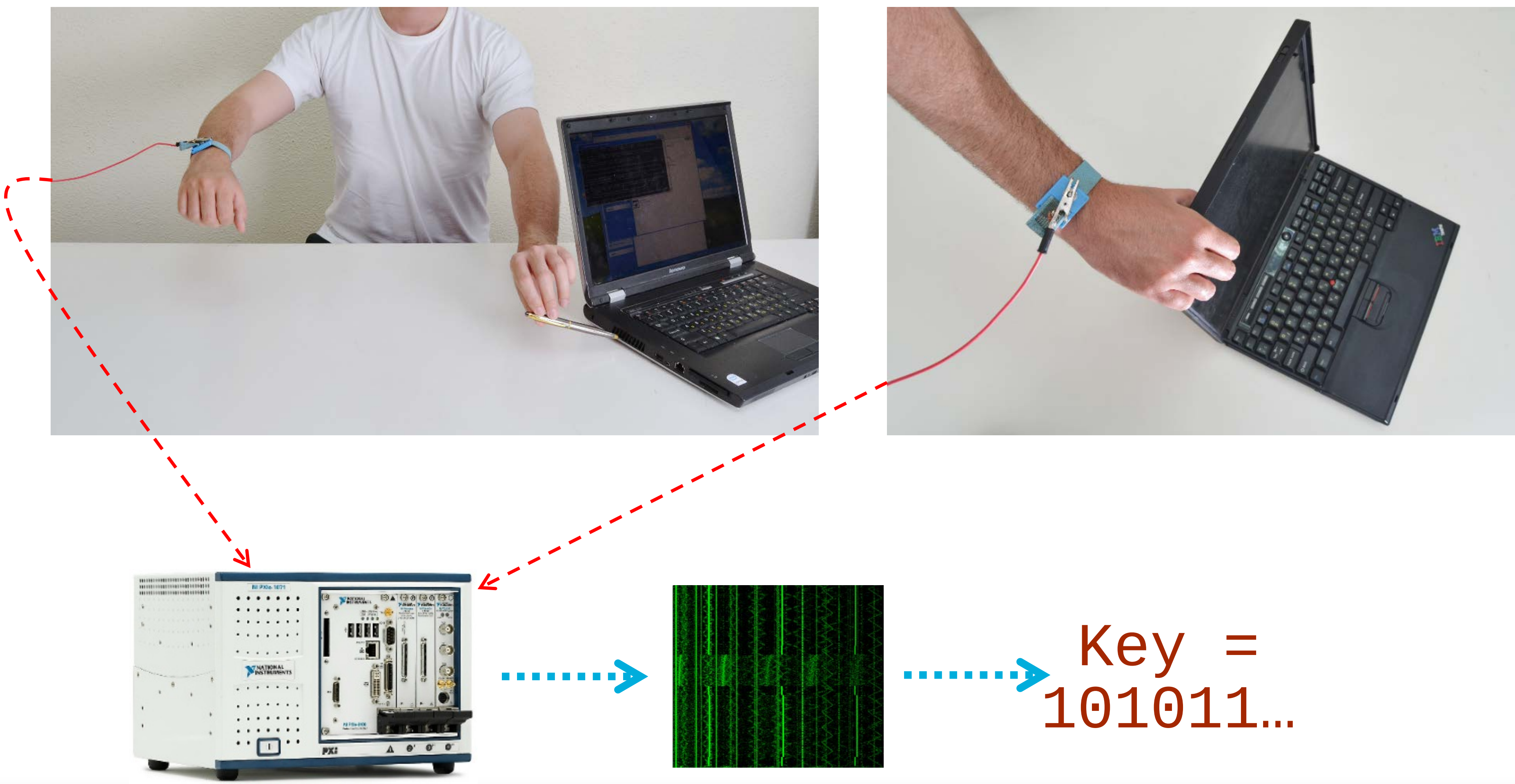


Reading the secret key (non-adaptive attack)

- Acquire trace
- Filter around carrier (1.7 MHz)
- FM demodulation
- Read out bits (“simple ground analysis”)

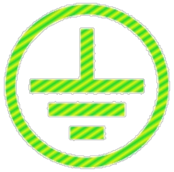


RSA and ElGamal key extraction in a few seconds using <u>human touch</u> (non-adaptive attack)



Ground-potential analysis

- **Attenuating EMI emanations**
  "Unwanted currents or electromagnetic fields? Dump them to the circuit ground!"
  (Bypass capacitors, RF shields, …)

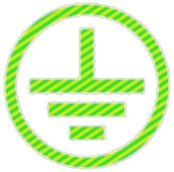


- Device is grounded, but its "ground" potential fluctuates relative to the mains earth ground.

	Computation
affects	currents and EM fields
dumped to	device ground
connected to	conductive chassis

Ground-potential analysis

- **Attenuating EMI emanations**
  "Unwanted currents or electromagnetic fields? Dump them to the circuit ground!"
  (Bypass capacitors, RF shields, …)

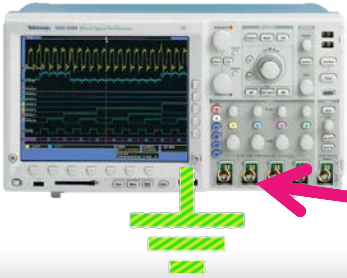


- Device is grounded, but its "ground" potential fluctuates relative to the mains earth ground.

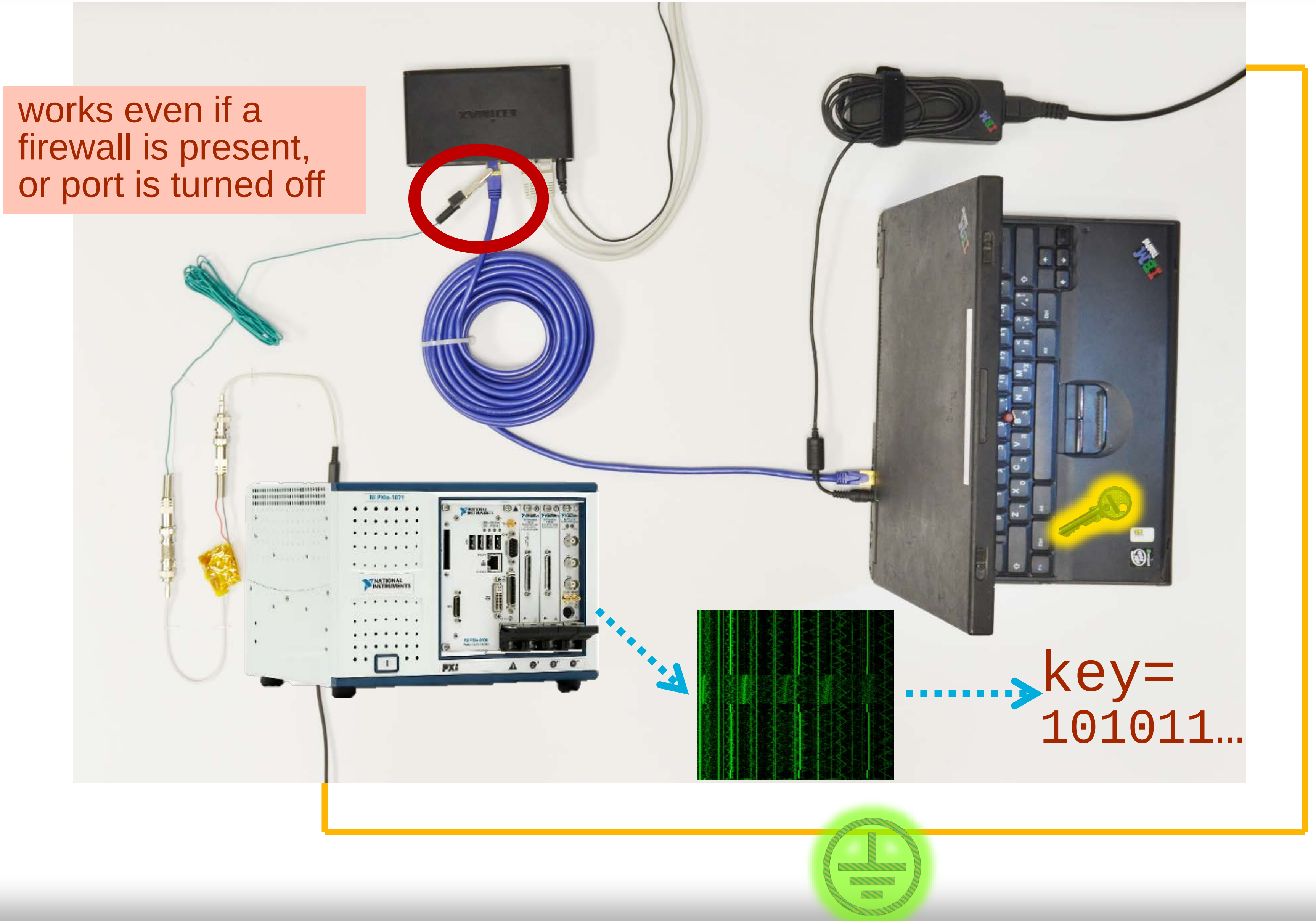
	Computation
affects	currents and EM fields
dumped to	device ground
connected to	conductive chassis
connected to	shielded cables

Even when no data, or port is turned off.

Key = ⟵····
101011…



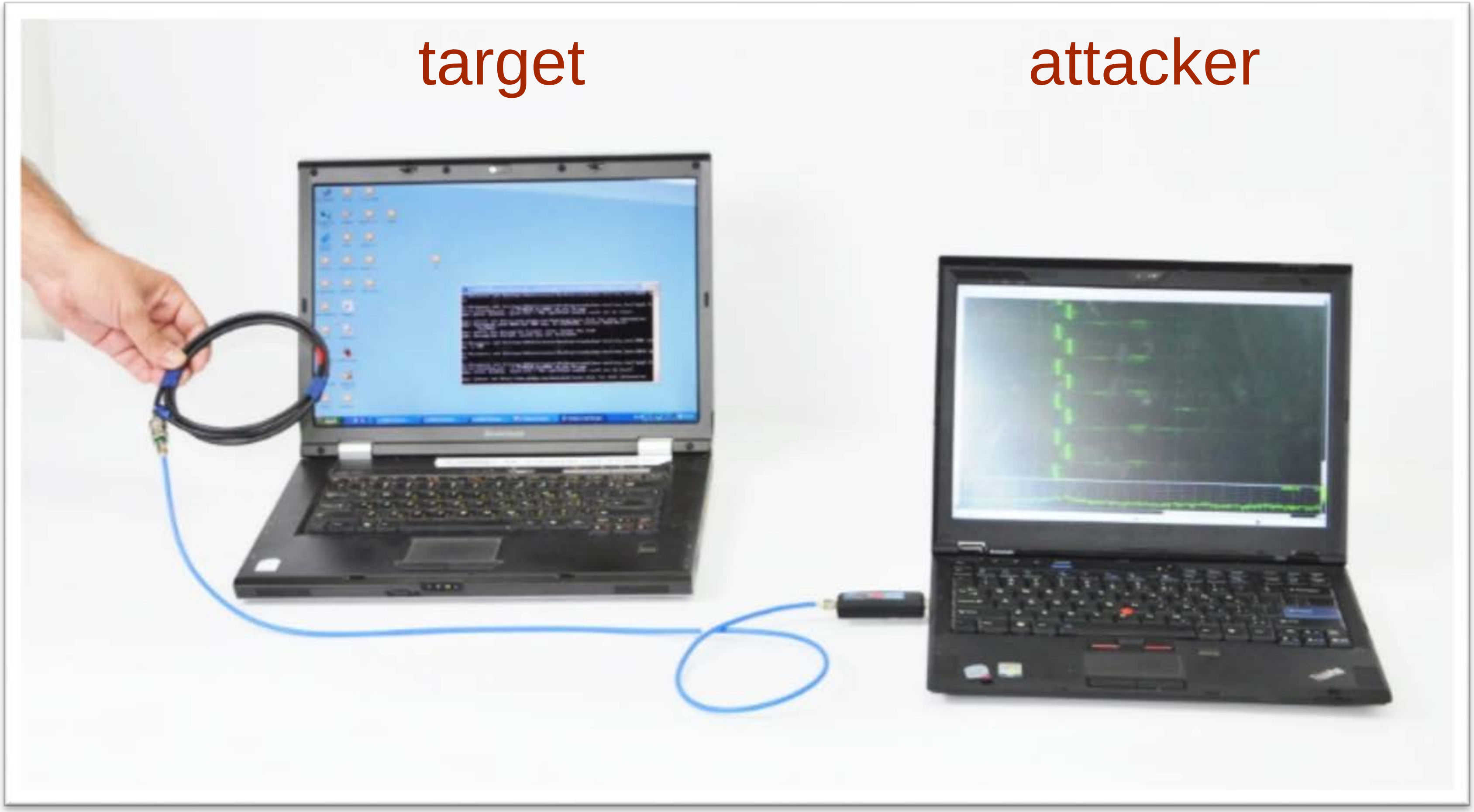
RSA and ElGamal key extraction in a few seconds using the far end of 10 meter network cable



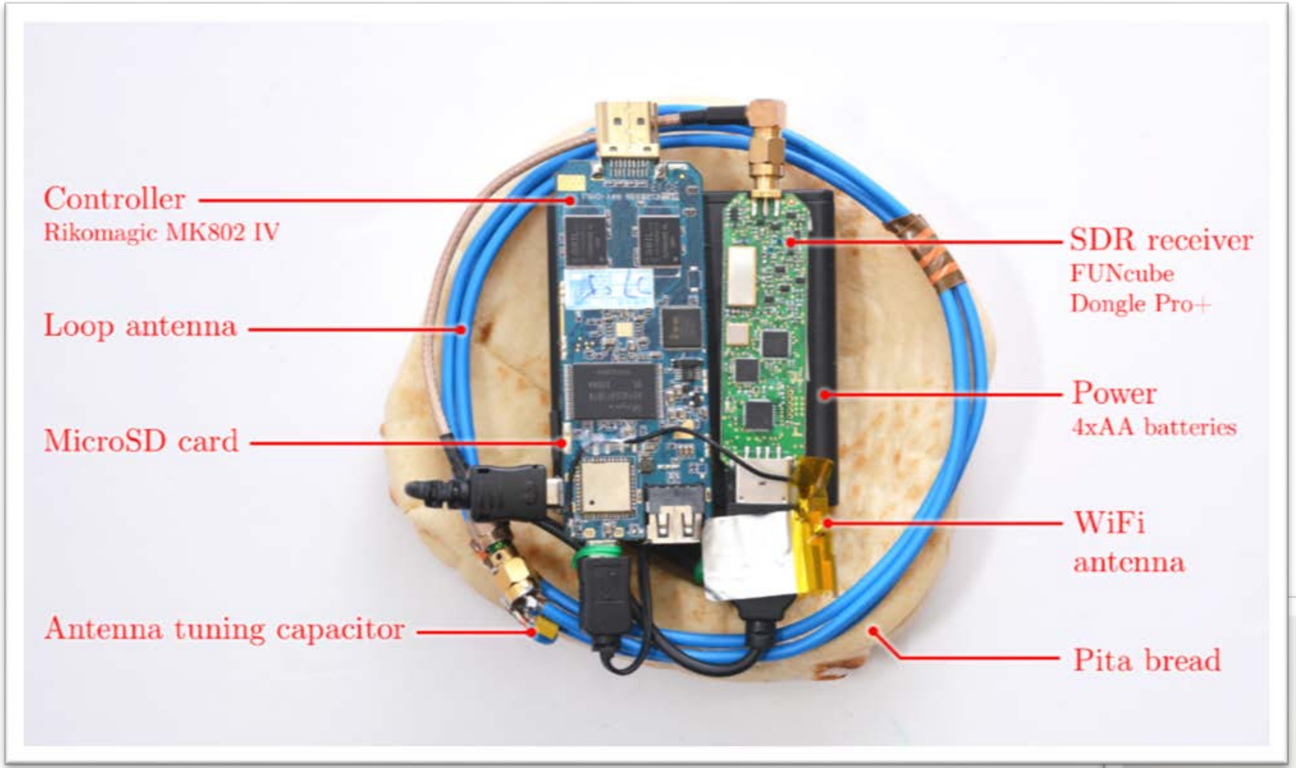
Empirical results: electromagnetic attacks

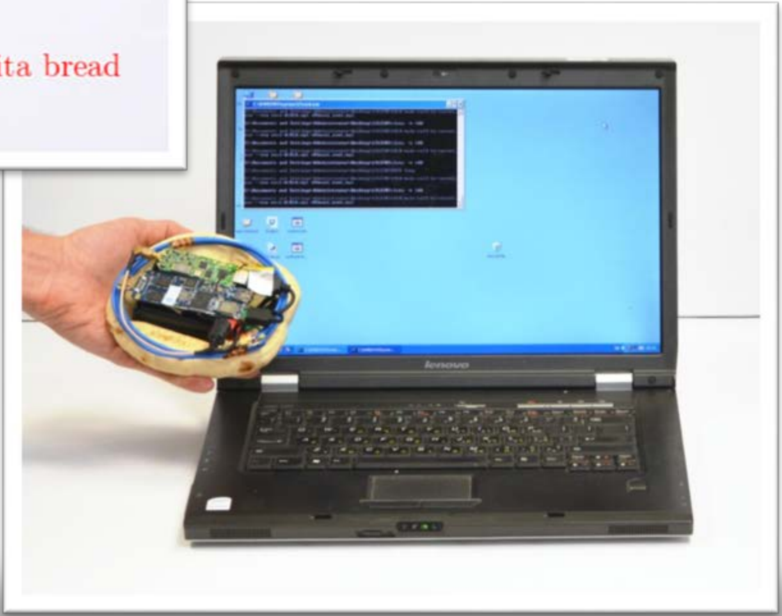
Electromagnetic key extraction

- Currents inside the target create electromagnetic waves.
- Can be detected using an electromagnetic probe (e.g., a loop of cable).



Portable Instrument for Trace Acquisition

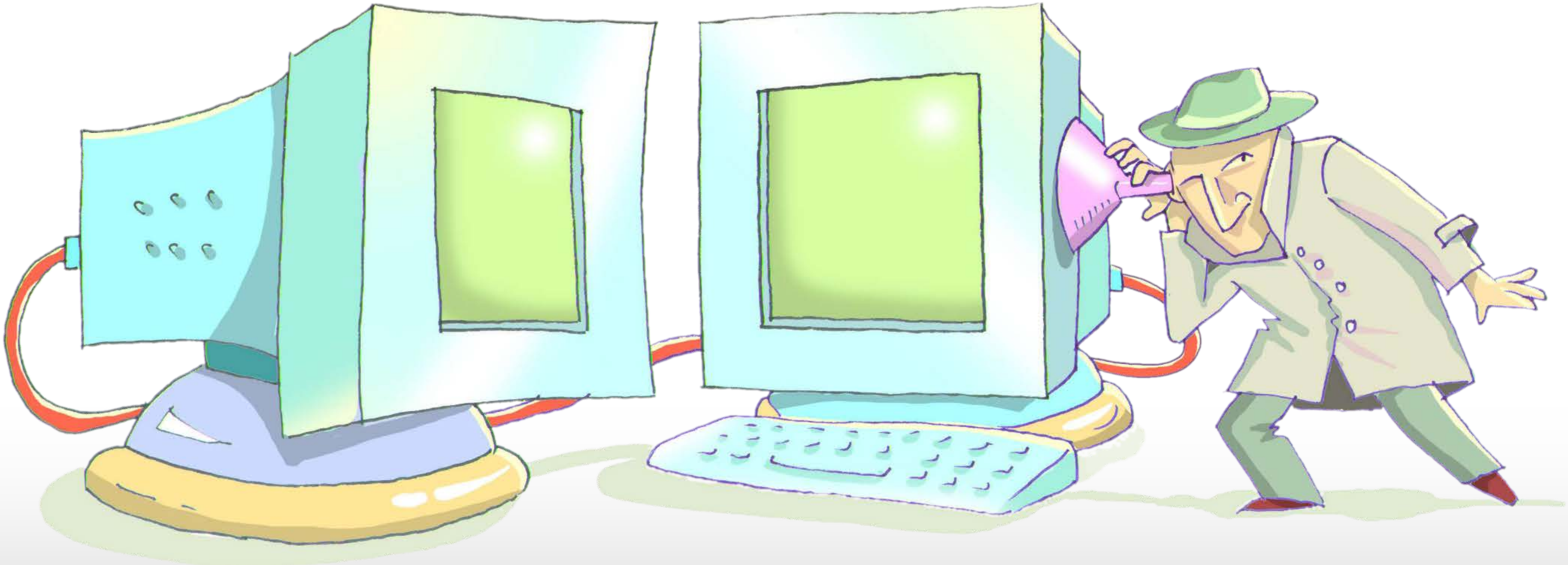




Cost to build: ~300$

Key extraction via commodity radio receiver

Acoustic cryptanalysis



Acoustic emanations from PCs

- Noisy electrical components in the voltage regulator

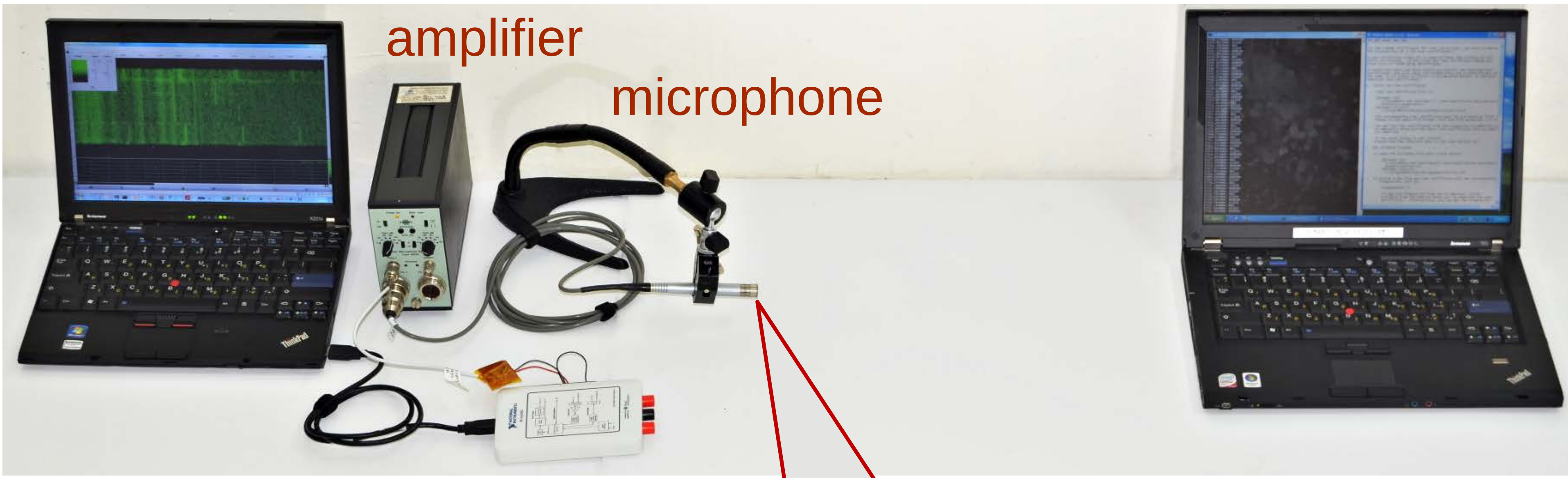
Bzzzzzz

- Commonly known as "coil-whine" but also originates from capacitors

Experimental setup (example)

attacker

target



digitizer

Adaptive key extraction

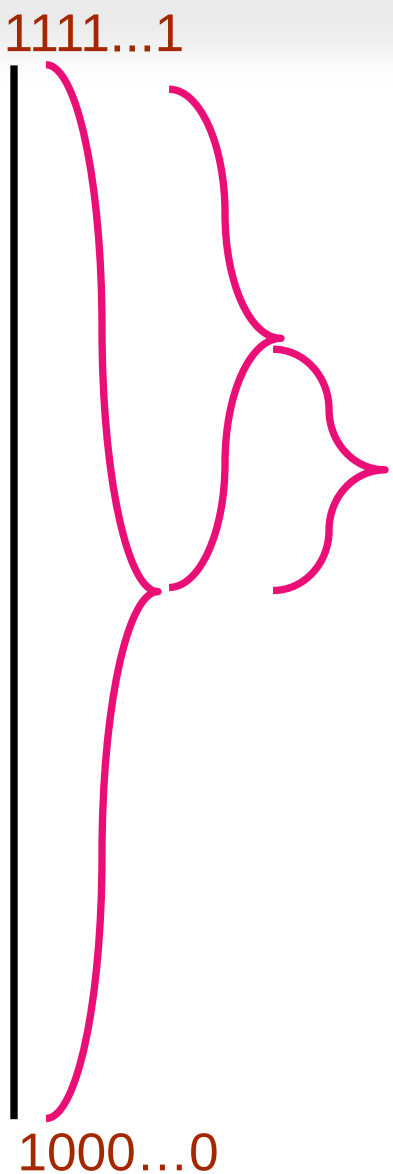
Severe attenuation of high frequency signals.

- Effective bandwidth of 50 kHZ
- Cannot observe a single squaring

Make the entire decryption depend on a single attacked bit

- Extreme version of self-amplification
- Extract the prime q bit-by-bit (adaptive chosen ciphertext)
- Total #measurements:
  2048 decryptions for RSA-4096 (~1 hour)

An adaptive chosen-ciphertext attack



$q = 11??????\ldots$

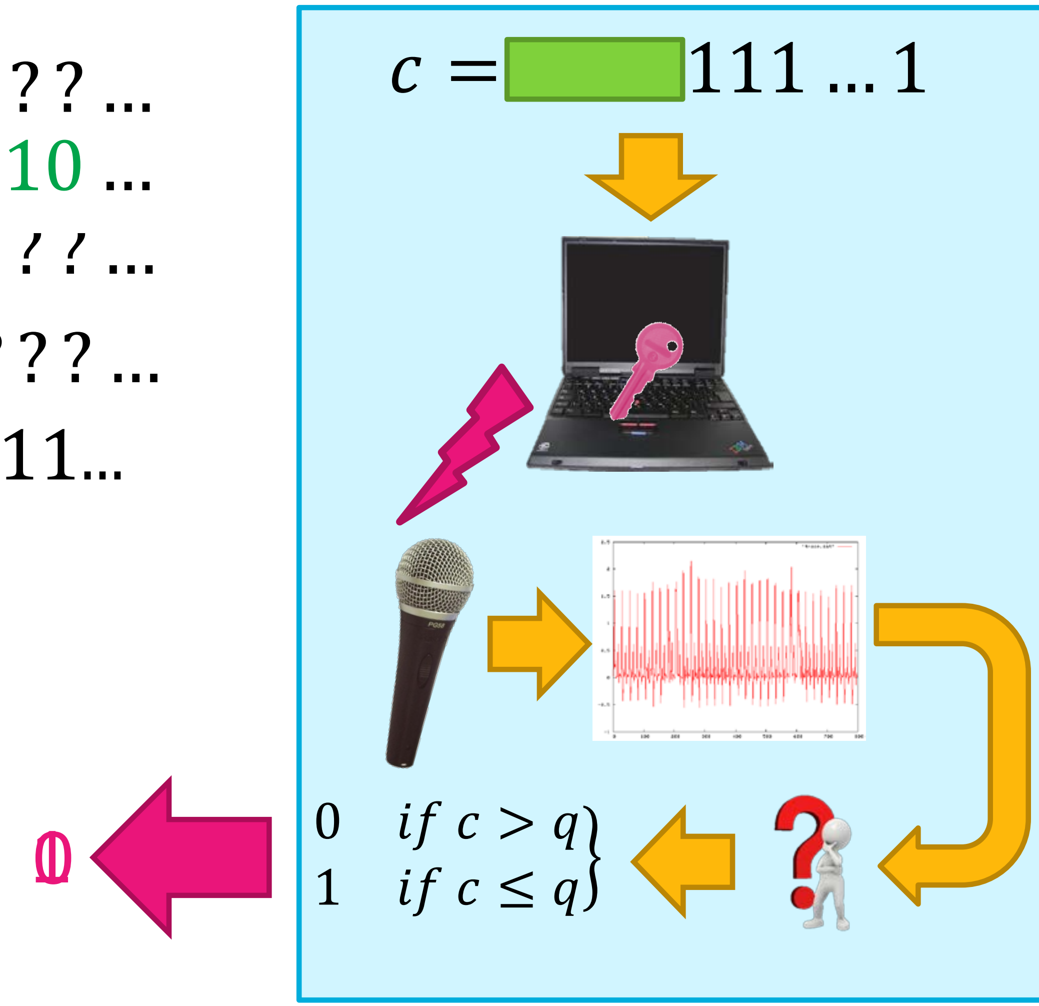
$q = 11011010\ldots$

$q = 110??????\ldots$

$q = 1????????\ldots$

$c = 10111111\ldots$

Bit-distinguisher oracle



An adaptive chosen-ciphertext attack

Total #measurements:

$$\frac{Key\ size}{2 \cdot 2} \cdot 2$$

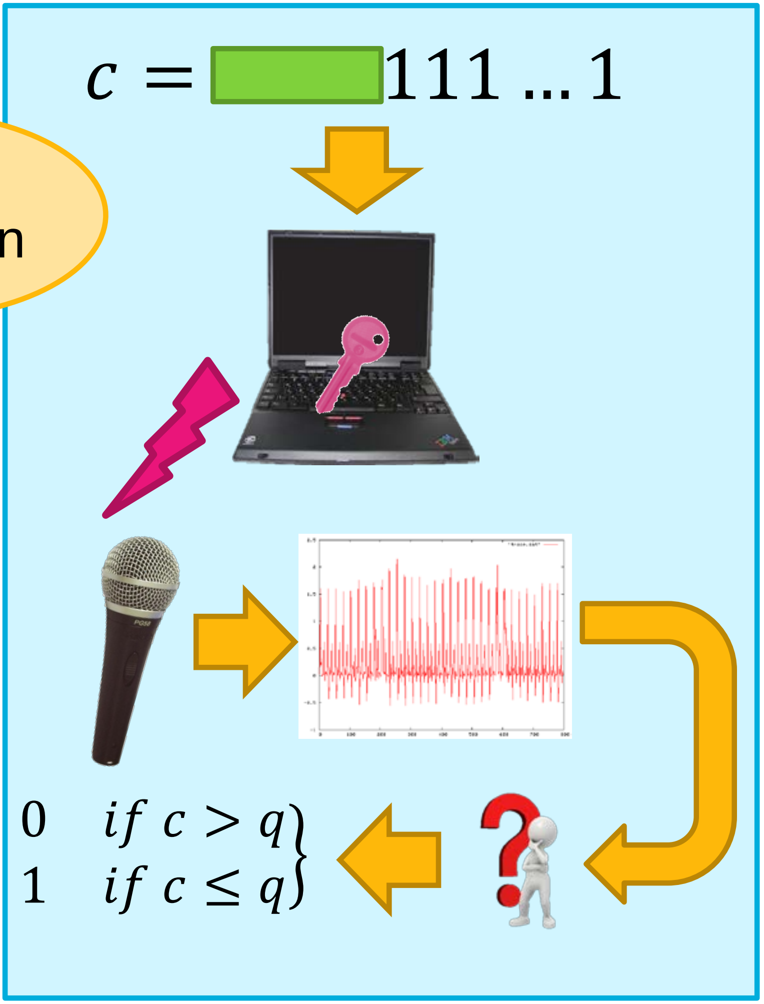
Error correction

Just *q*

Coppersmith lattice reduction: half the bits suffice

Overall: 2048 decryptions for RSA-4096 (~1 hour)

Bit distinguisher oracle



GnuPG RSA decryption - $m_q = c^{d_q} \, mod \, q$

```
modular_exponentiation(c,d,q){
  …
  karatsuba_mult(m,c)
  …
    karatsuba_mult(m,c){
      …
      basic_mult(x,y)
      …
        basic_mult(x,y){
        …
          if (y[j]==0)
            return 0
          else
            return y[j]*x
        }
    }
}
```

x7

x19

x2048

Grand total:
272384 times

~0.5 sec of measurements

craft c such that

$q_i = 1 \rightarrow y[j] = 0$

$q_i = 0 \rightarrow y[j] \neq 0$

(for most j's)

Extracting q_i (simplified)

GnuPG's modular exponentiation (see function mpi_powm in mpi/mpi-pow.c).

Input: Three integers c, d and q in binary representation such that $d = d_n \cdots d_1$.
Output: $m = c^d \mod q$.

```
procedure MODULAR_EXPONENTIATION(c, d, q)
    ~~if SIZE_IN_LIMBS(c) > SIZE_IN_LIMBS(q) then~~
        c ← c mod q
    m ← 1
    for i ← n downto 1 do
        m ← m^2
        if SIZE_IN_LIMBS(m) > SIZE_IN_LIMBS(q) then
            m ← m mod q
        if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then      ▷ defined as 16
            t ← MUL_BASECASE(m, c)                          ▷ Compute t ← m · c
        else
            t ← MUL(m, c)                                   ▷ Compute t ← m · c
        if SIZE_IN_LIMBS(t) > SIZE_IN_LIMBS(q) then
            t ← t mod q
        if d_i = 1 then
            m ← t
    return m
end procedure
```

$c^i = q_{2048} \cdots q_{i+1} 0 1 \cdots 1$

If $q_i = 1$ then $c^i < q$,
thus $c = c^i$.
That is, c has special structure.

If $q_i = 0$ then $2q > c^i > q$,
thus $c = c^i - q$.
That is, c is random looking.

and we now multiply by c causing the bit-dependent leakage.

Extracting q_i

GnuPG's modular exponentiation (see function mpi_powm in mpi/mpi-pow.c).

Input: Three integers c, d and q in binary representation such that $d = d_n \cdots d_1$.

Output: $m = c^d \mod q$.

```
procedure MODULAR_EXPONENTIATION(c, d, q)
    if SIZE_IN_LIMBS(c) > SIZE_IN_LIMBS(q) then
        c ← c mod q
    m ← 1
    for i ← n downto 1 do
        m ← m^2
        if SIZE_IN_LIMBS(m) > SIZE_IN_LIMBS(q) then
            m ← m mod q
        if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then      ▷ defined as 16
            t ← MUL_BASECASE(m, c)                          ▷ Compute t ← m · c
        else
            t ← MUL(m, c)                                   ▷ Compute t ← m · c
        if SIZE_IN_LIMBS(t) > SIZE_IN_LIMBS(q) then
            t ← t mod q
        if d_i = 1 then
            m ← t
    return m
end procedure
```

$c^i = q_{2048} \cdots q_{i+1} 01 \cdots 1 + n$

If $q_i = 1$ then $c^i - n < q$, thus $c = c^i - n$.
That is, c has special structure.

If $q_i = 0$ then 2q $> c^i - n > q$, thus $c = c^i - q - n$.
That is, c is random looking.

and we now multiply by c causing the bit-dependent leakage.

Extracting q_i (problem)

GnuPG's modular exponentiation (see function mpi_powm in mpi/mpi-pow.c).

Input: Three integers c, d and q in binary representation such that $d = d_n \cdots d_1$.

Output: $m = c^d \mod q$.

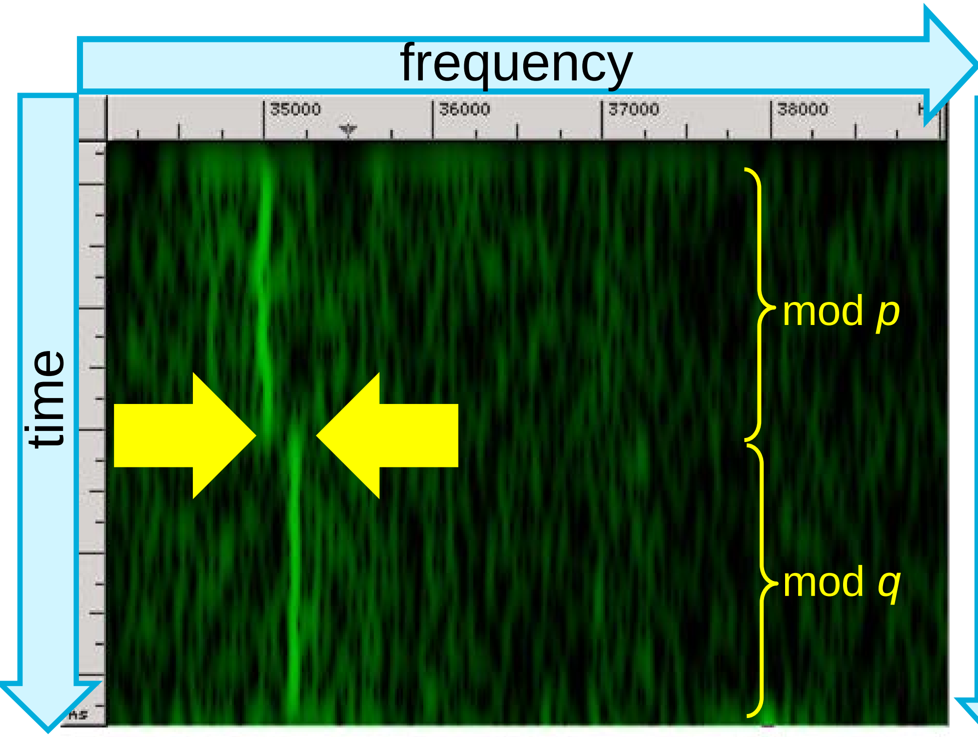
```
procedure MODULAR_EXPONENTIATION(c, d, q)
    if SIZE_IN_LIMBS(c) > SIZE_IN_LIMBS(q) then
        c ← c mod q
    m ← 1
    for i ← n downto 1 do
        m ← m^2
        if SIZE_IN_LIMBS(m) > SIZE_IN_LIMBS(q) then
            m ← m mod q
        if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then        ▷ defined as 16
            t ← MUL_BASECASE(m, c)                            ▷ Compute t ← m · c
        else
            t ← MUL(m, c)                                     ▷ Compute t ← m · c
        if SIZE_IN_LIMBS(t) > SIZE_IN_LIMBS(q) then
            t ← t mod q
        if d_i = 1 then
            m ← t
    return m
end procedure
```

Multiplication is repeated 2048 times (0.5 sec of data)

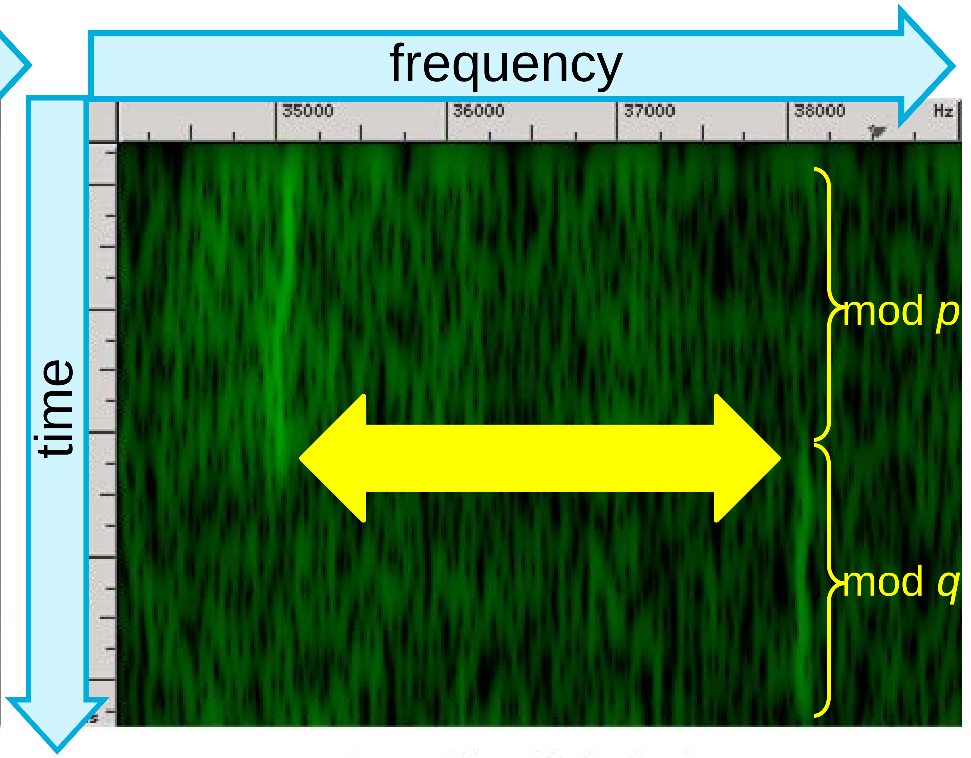
Single multiplication is way too fast for us to measure

Empirical results: acoustic attacks

Distinguishing a key bit by a spectral signature



Attacked bit is 0



Attacked bit is 1

Demo:
key extraction

Acoustic: results

RSA 4096-bit key extraction from
1 meter away using a microphone

Acoustic: results

RSA 4096-bit key extraction from

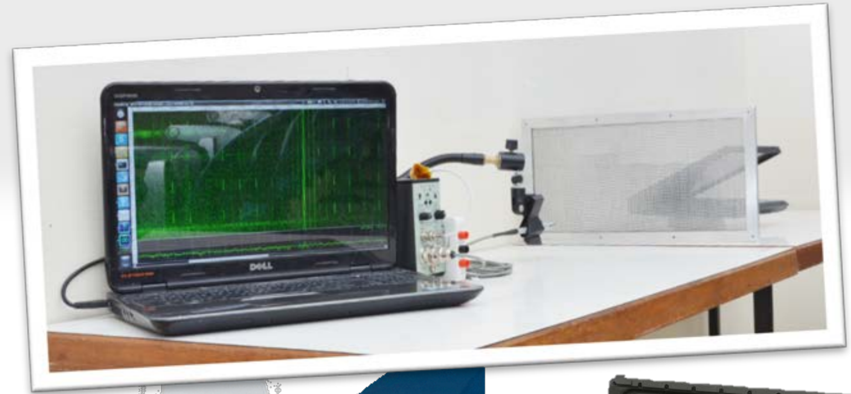
10 meters away using a parabolic microphone

Acoustic: results

RSA 4096-bit key extraction from
30cm away using a **smartphone**

Countermeasures

Countermeasures



Common suggestions

1. Shielding
   - EM (Faraday cages), ground
     difficult and expensive
   - Acoustic? Vents!

2. Add analog noise (expensive, correlations remain)
3. Parallel software load (inadequate, may *help* attacker)

Attacks rely on decryption of chosen ciphertexts.

Solution: ciphertext randomization use equivalent but random-looking ciphertexts

- Negligible slowdown for RSA
- x2 slowdown for ElGamal

Effective countermeasure: ciphertext randomization (added in GnuPG 1.4.16)

Given a ciphertext c:

1. Generate a random number r and compute r^e
2. Decrypt $r^e \cdot c$ and obtain m'
3. Output $m' \cdot r^{-1}$

Works since $ed = 1 \; mod \; \varphi(n)$ thus:

$$
\begin{aligned}
(r^e \cdot c)^d \cdot r^{-1} \; mod \; n &= r^{ed} \cdot r^{-1} \cdot c^d \; mod \; n \\
&= r \cdot r^{-1} \cdot c^d \; mod \; n \\
&= c^d \; mod \; n \\
&= m
\end{aligned}
$$

tau.ac.il/~tromer/acoustic	CRYPTO'14	CVE 2013-4576
tau.ac.il/~tromer/handsoff	CHES'14	CVE-2014-5270
tau.ac.il/~tromer/radioexp	CHES'15	CVE-2014-3591