RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis

Matteo Berti

matteo.berti11@studio.unibo.it

LM Informatica

A.A. 2018/2019

Despite there are **many side-channels attacks** (electromagnetic, power-monitoring, timing, optical, acoustic, …), **this research** is interesting because it is the **only** available **source** on acoustic cryptanalysis **of a cryptosystem**.

We will cover the following sections:

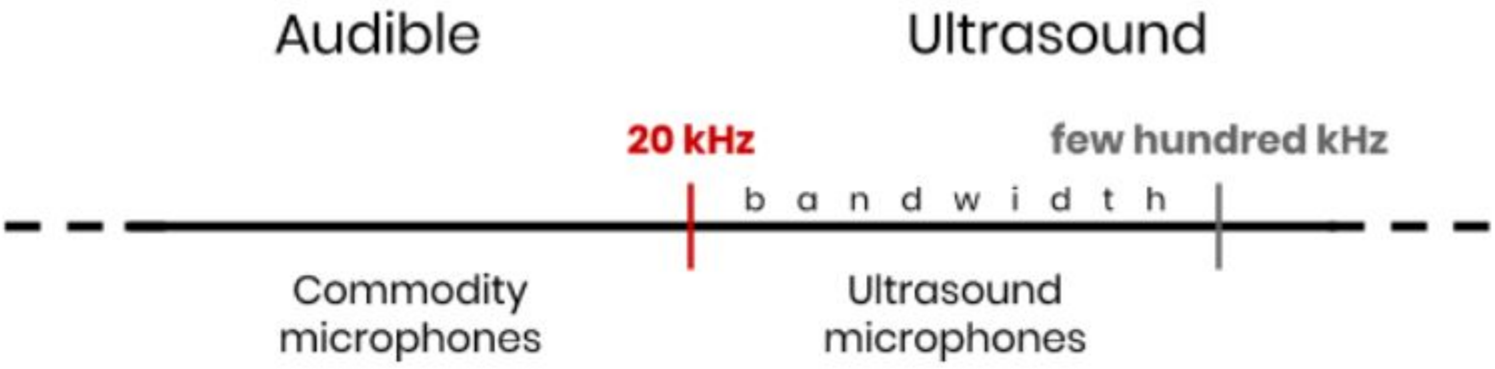
- **Introduction**
- **Foundations of the attack**
- **Further detail on the cryptanalysis**
- **Problems**
- **Error detection**
- **Attack mitigation**

Introduction

CPUs change power according to the **type of operations** they perform.

Electronic components in the computers **generate vibrations**.

The **bandwidth** of these signals is **very low**:



Introduction

GnuPG **operations** can be **identified** by their **acoustic frequency spectrum**.

↓

GPG RSA **secret keys** can be **distinguished** by the **sound** they made.

↓

Therefore, the attack requires **ciphertexts** adaptively **chosen by** the **attacker**:

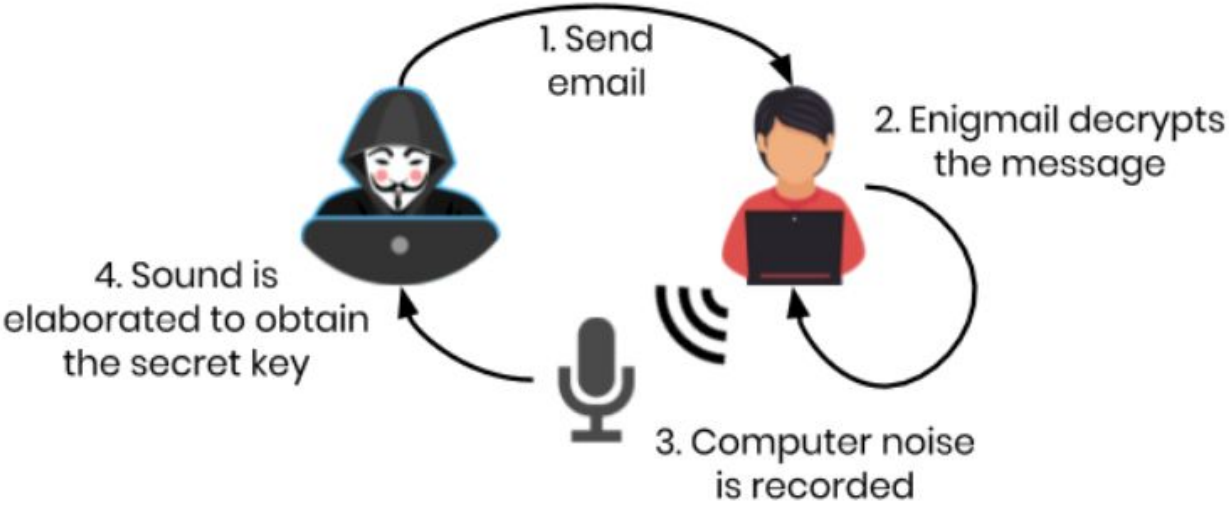
Chosen-ciphertext channel by email.

Introduction

A suitable ciphertext **attack vector** is:

OpenPGP encrypted email messages.

Enigmail: Thunderbird **plugin** that **automatically decrypts incoming** email for notification purposes.


1. Send email
2. Enigmail decrypts the message
3. Computer noise is recorded
4. Sound is elaborated to obtain the secret key

Introduction

Other ways to eavesdrop secret keys:

- A **mobile device** remotely **compromised**, which record the target computer noise.

- The **target computer** if compromised may **spy on itself**.

Introduction

Three levels of recording accuracy:

- **Lab-grade setup**: **Brüel&Kjær condenser microphones** with 3 capsules (350kHz, 40kHz, 21kHz):

- **Portable setup**: same **Brüel&Kjær capsules** as before but replaced some components to **fit in a briefcase** (100kHz).

Introduction

Three levels of recording accuracy:

- **Mobile-phone setup**: were used several **Android smartphones** (24kHz).

Distant acquisition:

- **Parabolic microphones**: increase effective range from 1 meter to **4 meter**.

Foundations of the attack

Recall on RSA cryptosystem:

- 2 large random primes ***p*** and ***q***
- 2 numbers ***e*** and ***d*** such that $\boldsymbol{ed = 1 \bmod \varphi(n)}$ and $\boldsymbol{n = pq}$

Encryption: $\boldsymbol{m^e \bmod n}$ *Decryption*: $\boldsymbol{c^d \bmod n}$

public key

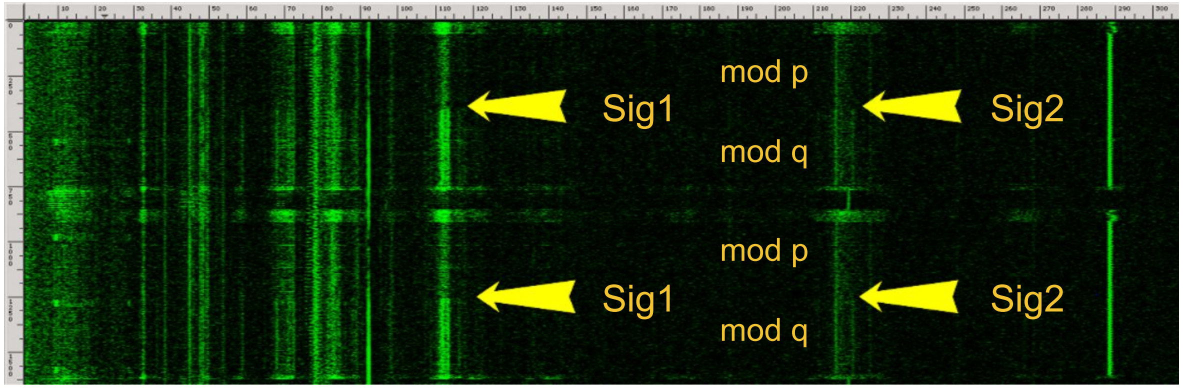
pk = (n, e)

secret key

sk = (d, p, q)

The ***signature*** is computed:

$$\begin{tikzcd} m^{d \bmod (p-1)} \bmod p \arrow[dr] & & m^{d \bmod (q-1)} \bmod q \arrow[dl] \\ & s = m^d \bmod n & \end{tikzcd}$$

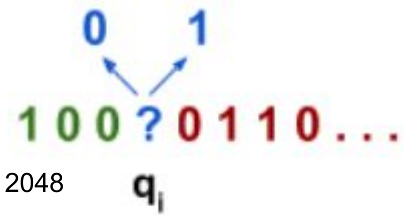


Each signature has a unique spectral signature (2 signatures and 4 modules above).

Foundations of the attack

The attack exposes the **secret factor q one bit at a time**, from MSB to LSB.

For each bit q_i we **assume** that $\mathbf{q_{2048} \ldots q_{i+1}}$ were correctly **recovered**, and **check** if $\mathbf{q_i}$ is **0 or 1**.



Eventually, we **learn all of q** and recover the factorization of n.

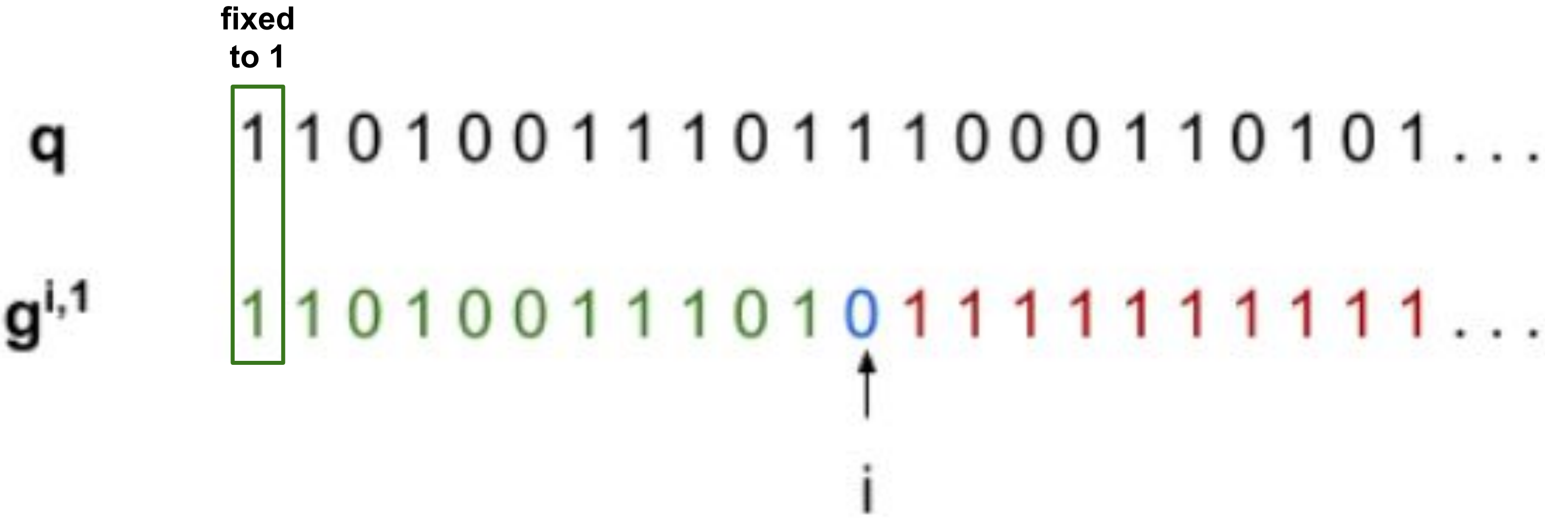
q 1 0 0 1 0 1 1 0 . . .
p ? ? ? ? ? ? ? ? . . .
n / q = p

The same technique applies to p, but **q** has a **better signal**.

Foundations of the attack

Let $\mathbf{g^{i,1}}$ be the **ciphertext** whose **topmost i−1 bits** are correctly **recovered from q**, the **i-th bit** is **0**, and the **remaining** (low) **bits** are **1**.

Moreover **RSA keys** in GPG have **MSB** of **q** is **set**: $q_{2048} = 1$



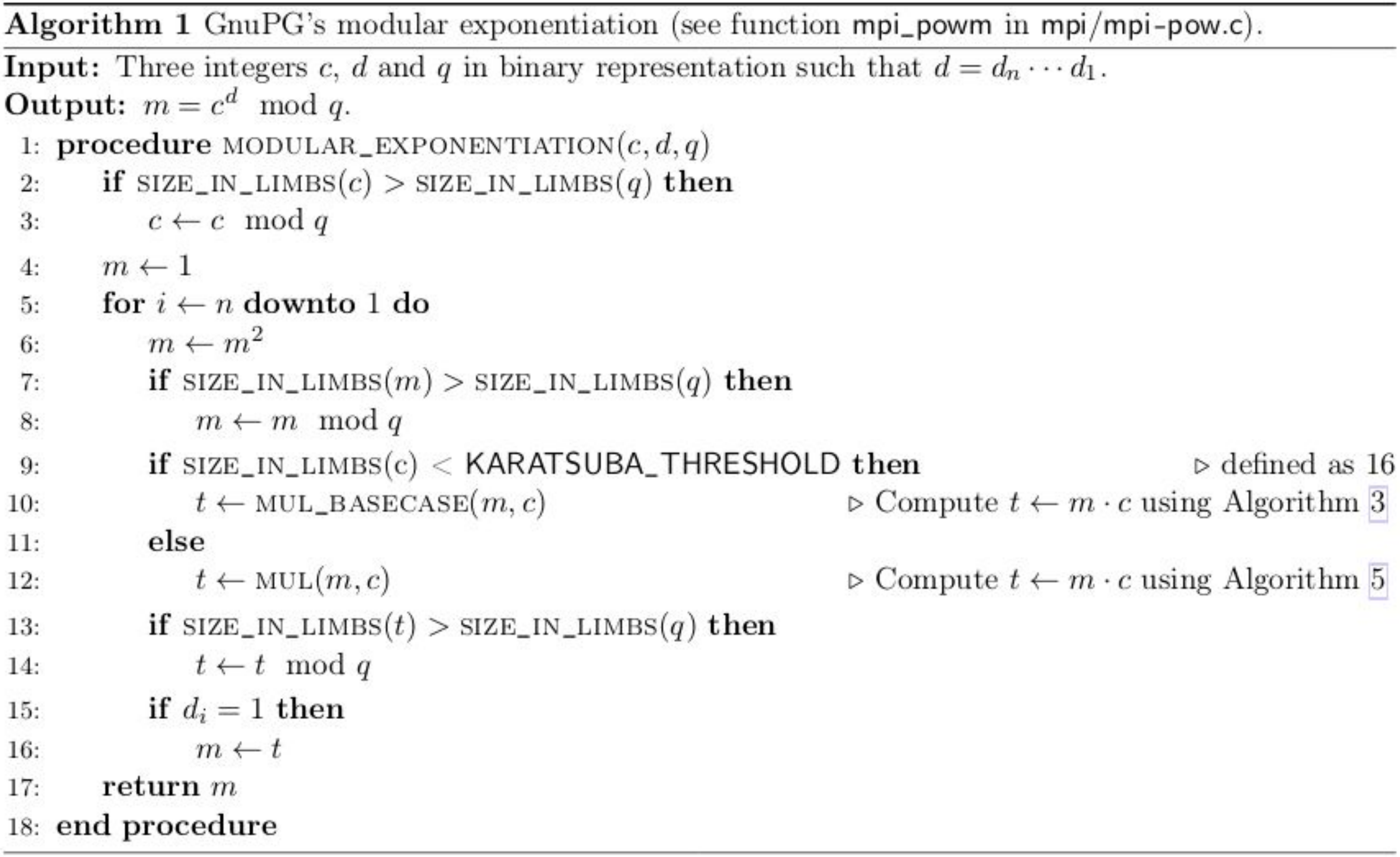
fixed to 1

q 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0 0 1 1 0 1 0 1 . . .

$g^{i,1}$ 1 1 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 . . .

i

Foundations of the attack



Algorithm 1 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
        $c \leftarrow c \mod q$
    $m \leftarrow 1$
    for $i \leftarrow n$ downto 1 do
        $m \leftarrow m^2$
        if SIZE_IN_LIMBS($m$) > SIZE_IN_LIMBS($q$) then
            $m \leftarrow m \mod q$
        if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then          ▷ defined as 16
            $t \leftarrow$ MUL_BASECASE($m, c$)              ▷ Compute $t \leftarrow m \cdot c$ using Algorithm 3
        else
            $t \leftarrow$ MUL($m, c$)                       ▷ Compute $t \leftarrow m \cdot c$ using Algorithm 5
        if SIZE_IN_LIMBS($t$) > SIZE_IN_LIMBS($q$) then
            $t \leftarrow t \mod q$
        if $d_i = 1$ then
            $m \leftarrow t$
    return $m$
end procedure
```

A ***limb*** is the part of a multi-precision number that fits in a **single machine word**, normally a limb is **32** or **64 bits**.

Foundations of the attack

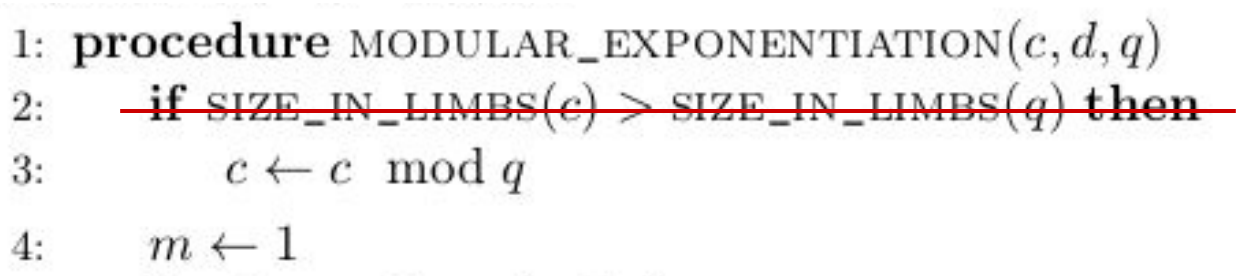
When we **decrypt $g^{i,1}$**, i-th bit of q could be:

- **q_i = 1** then $g^{i,1} < q$

q	1 1 0 1 0 0 1 1 = 211
$g^{i,1}$	1 1 0 0 1 1 1 1 = 207

If we assume **line 2** of Alg1 is **removed**.

- Line 3: **c ← c mod q** **returns c** because $c = g^{i,1} < q$



```
1: procedure MODULAR_EXPONENTIATION($(c, d, q)$)
2:     ~~if SIZE_IN_LIMBS($(c)$) $>$ SIZE_IN_LIMBS($(q)$) then~~
3:         $c \leftarrow c \mod q$
4:     $m \leftarrow 1$
```

Foundations of the attack

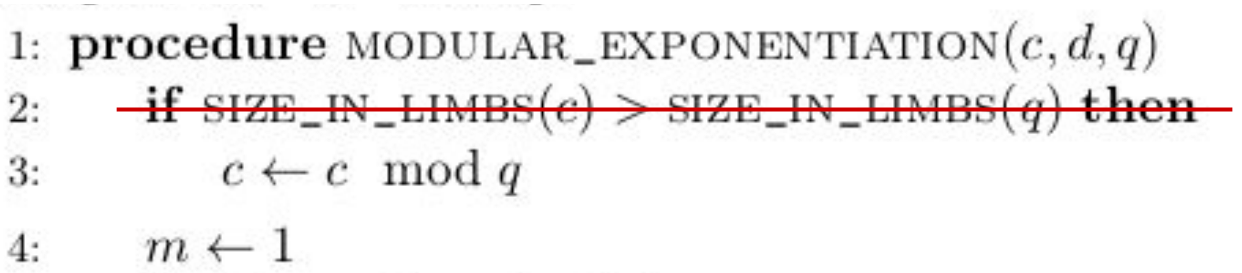
When we **decrypt $g^{i,1}$**, i-th bit of q could be:

- **q_i = 0** then $g^{i,1} \geq q$

q	1 1 0 0 1 0 0 1 = 201
$g^{i,1}$	1 1 0 0 1 1 1 1 = 207

If we assume **line 2** of Alg1 is **removed**.

- Line 3: **c ← c mod q** **returns c − q** because $q \leq g^{i,1} < 2q$



```
1: procedure MODULAR_EXPONENTIATION($c, d, q$)
2:     ~~if SIZE_IN_LIMBS($c$) > SIZE_IN_LIMBS($q$) then~~
3:         $c \leftarrow c \mod q$
4:     $m \leftarrow 1$
```

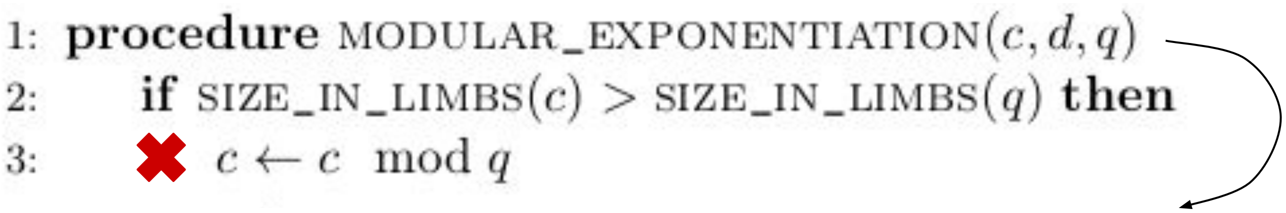
The occurrence or not of **this reduction** will lead us to **distinguish** if the **bit** of **q** is **1** or **0**.

Foundations of the attack

If we enable again line 2 of Alg1, we see **line 3** is **never taken**.

This happens because **g^{i,1}** and **q** have the **same number of limbs** (64 each).

```
SIZE_IN_LIMS(c) > SIZE_IN_LIMBS(q) → 64 > 64 → FALSE
```



```
1: procedure MODULAR_EXPONENTIATION($c, d, q$)
2:     if SIZE_IN_LIMBS($c$) > SIZE_IN_LIMBS($q$) then
3:     ✖ $c \leftarrow c \mod q$
```

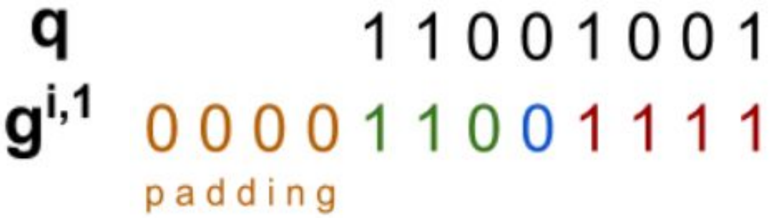
But **we need the reduction** to distinguish $q_i = 0$ from $q_i = 1$

This can be solved in either of two way.

Foundations of the attack

1. It could be **added leading zero limbs** to **$g^{i,1}$**, so **line 3** will be **always taken**.

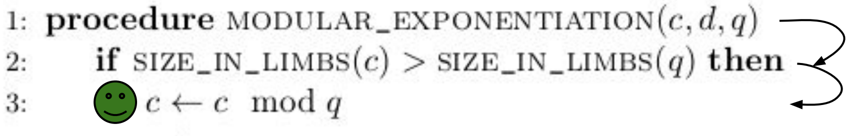
X But the **algorithm** could be **changed** to not allocate leading zero limb.



2. It could be **decrypted** the 128 limb **number $g^{i,1}$ + n** (the result would be the same) so **line 3** will be **always taken**.

! DECRYPT(**$g^{i,1}$ + n**)

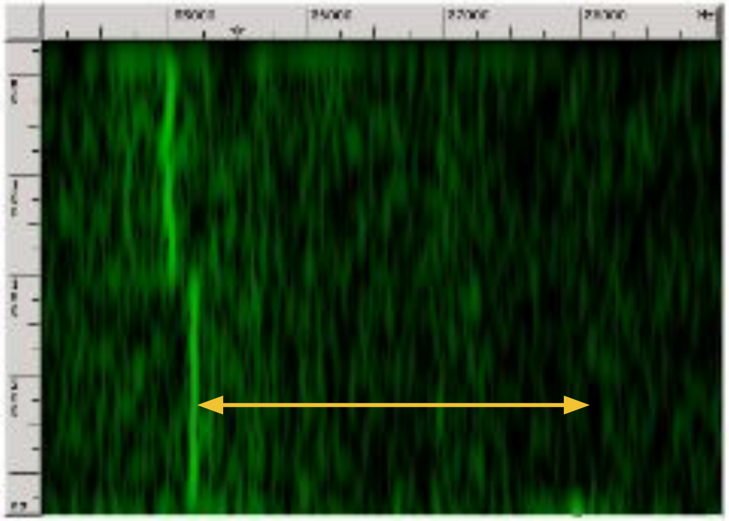
`#limbs(g^{i,1} + n)=128 > #limbs(q)=64`



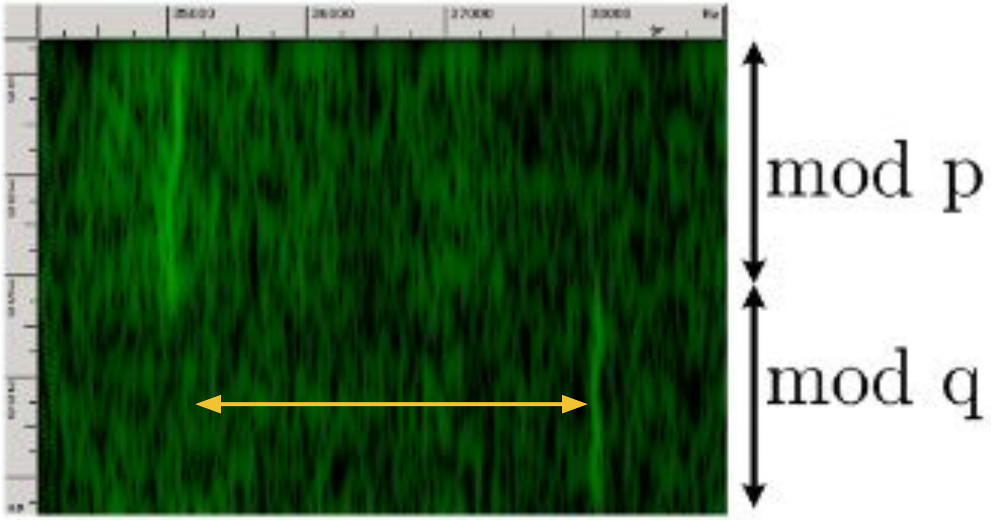
```
1: procedure MODULAR_EXPONENTIATION($c, d, q$)
2:     if SIZE_IN_LIMBS($c$) > SIZE_IN_LIMBS($q$) then
3:         $c \leftarrow c \mod q$
```

Foundations of the attack

As we can see in the figures when $\mathbf{q_i = 0}$ the **frequency** of the modular exponentiation is **lower than** when $\mathbf{q_i = 1}$.



Attacked bit is 0

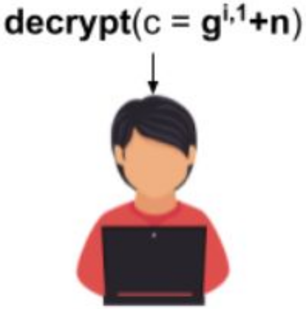


Attacked bit is 1

Foundations of the attack

To sum up what we have seen thus far:

1. **Decrypt c (= $g^{i,1}$ + n)** on the target machine.

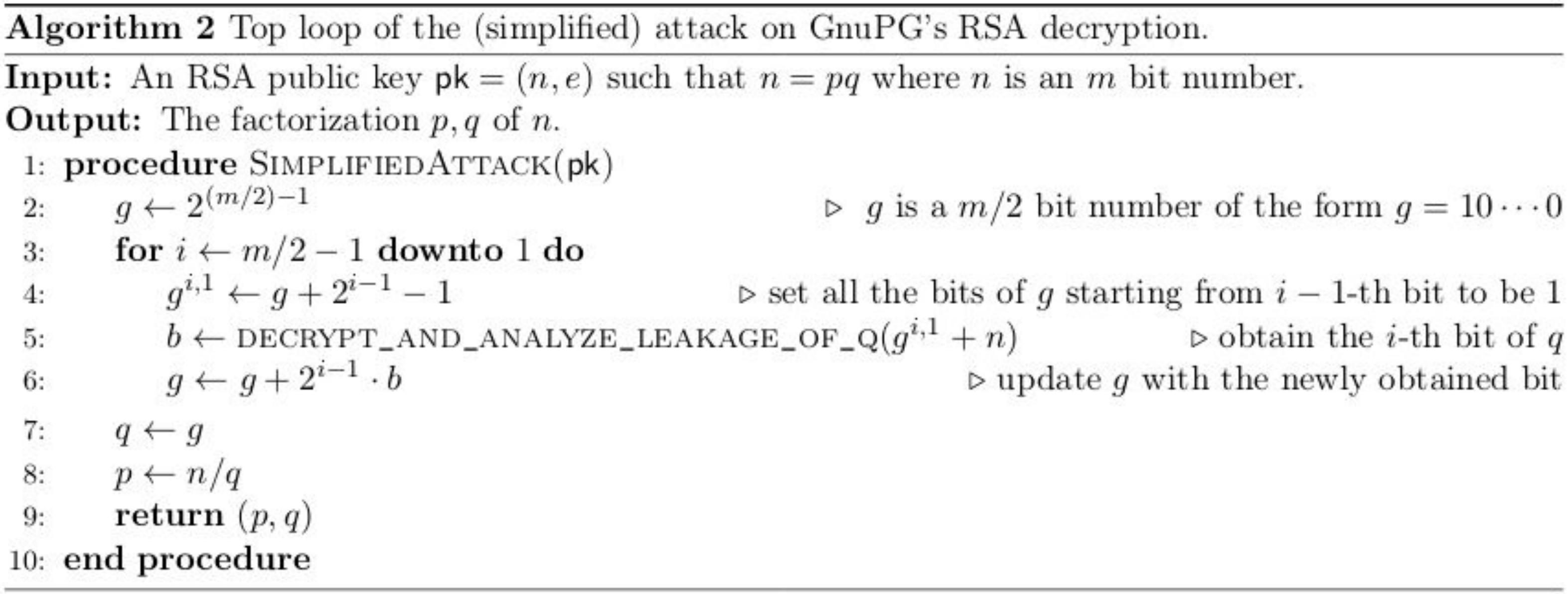


2. **Measure acoustic leakage** during decryption.

3. **Recognize** the difference between the **two leakage patterns**.

This can be done sending **email messages** with the **chosen ciphertext** backdated or marked as **spam**.

Foundations of the attack

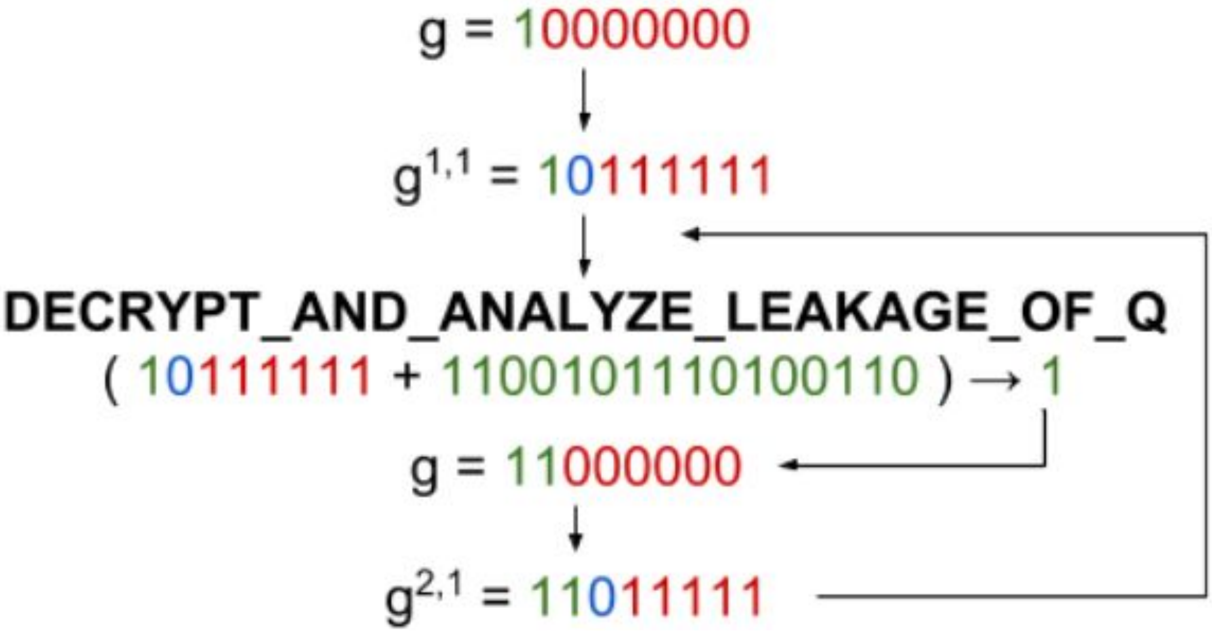


Algorithm 2 Top loop of the (simplified) attack on GnuPG's RSA decryption.

Input: An RSA public key $\mathsf{pk} = (n, e)$ such that $n = pq$ where n is an m bit number.

Output: The factorization p, q of n.

```
procedure SimplifiedAttack(pk)
    $g \leftarrow 2^{(m/2)-1}$                    ▷ $g$ is a $m/2$ bit number of the form $g = 10\cdots0$
    for $i \leftarrow m/2 - 1$ downto 1 do
        $g^{i,1} \leftarrow g + 2^{i-1} - 1$      ▷ set all the bits of $g$ starting from $i-1$-th bit to be 1
        $b \leftarrow$ DECRYPT_AND_ANALYZE_LEAKAGE_OF_Q($g^{i,1} + n$)      ▷ obtain the $i$-th bit of $q$
        $g \leftarrow g + 2^{i-1} \cdot b$         ▷ update $g$ with the newly obtained bit
    $q \leftarrow g$
    $p \leftarrow n/q$
    return $(p, q)$
end procedure
```



g = 10000000

$g^{1,1}$ = 10111111

DECRYPT_AND_ANALYZE_LEAKAGE_OF_Q

(10111111 + 1100101110100110) → 1

g = 11000000

$g^{2,1}$ = 11011111

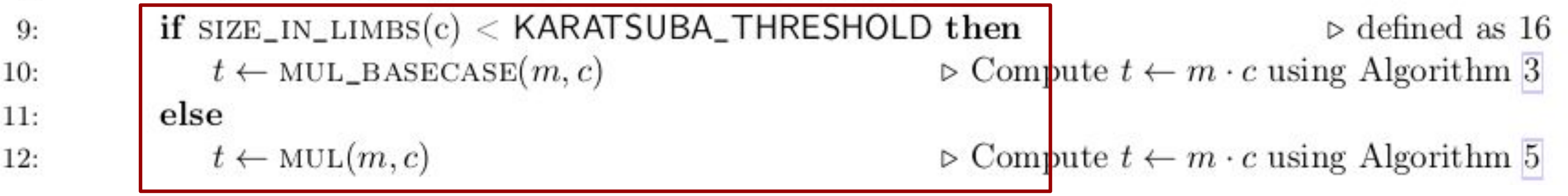
Further detail on the cryptanalysis

But exactly what makes the **difference** in the **acoustic frequency** when the bit attacked is 1 or 0?

To understand this we need to go **deeper** in the **modular exponentiation** algorithm.

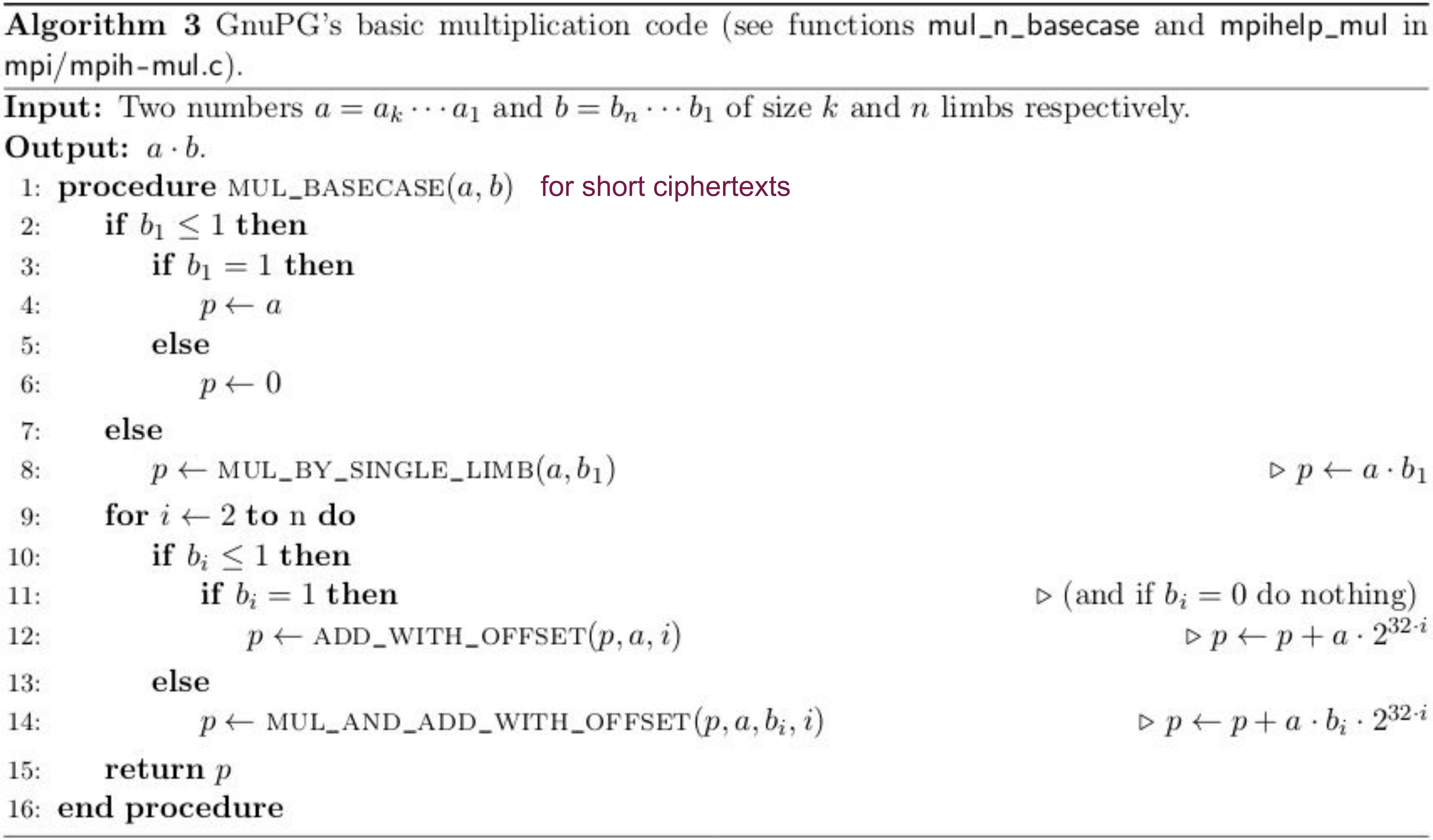
The algorithm consists of **two main** multiplication **routines**:
- A **basic schoolbook** multiplication routine (for short ciphertexts).
- A **recursive Karatsuba** multiplication algorithm (for large ciphertexts).

MODULAR_EXPONENTIATION



```
if SIZE_IN_LIMBS(c) < KARATSUBA_THRESHOLD then                ▷ defined as 16
    $t \leftarrow$ MUL_BASECASE($m, c$)          ▷ Compute $t \leftarrow m \cdot c$ using Algorithm 3
else
    $t \leftarrow$ MUL($m, c$)                   ▷ Compute $t \leftarrow m \cdot c$ using Algorithm 5
```

Further detail on the cryptanalysis



Algorithm 3 GnuPG's basic multiplication code (see functions mul_n_basecase and mpihelp_mul in mpi/mpih-mul.c).

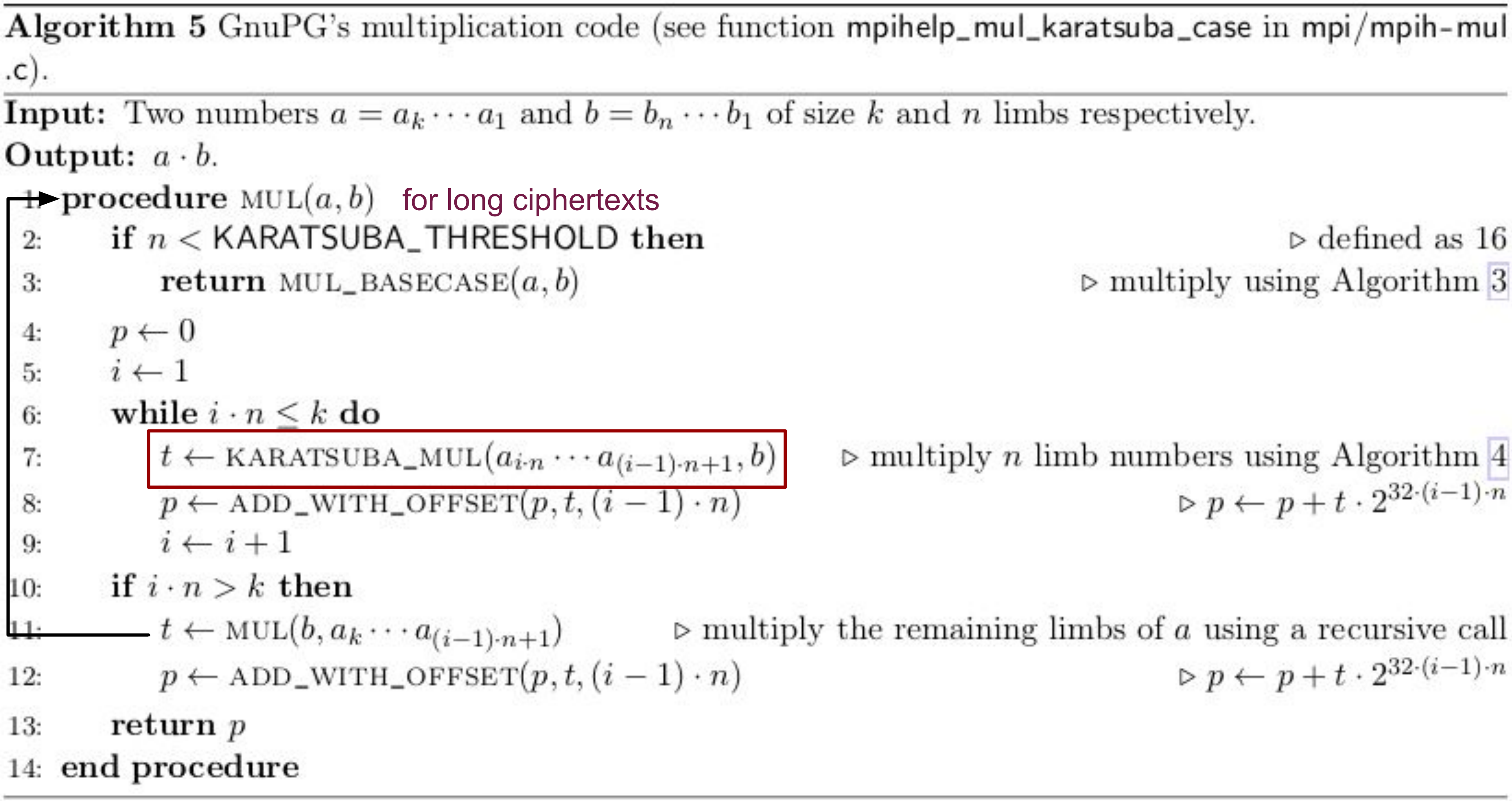
Input: Two numbers $a = a_k \cdots a_1$ and $b = b_n \cdots b_1$ of size k and n limbs respectively.

Output: $a \cdot b$.

```
procedure MUL_BASECASE($(a, b)$)
    if $b_1 \leq 1$ then
        if $b_1 = 1$ then
            $p \leftarrow a$
        else
            $p \leftarrow 0$
    else
        $p \leftarrow$ MUL_BY_SINGLE_LIMB$(a, b_1)$                         ▷ $p \leftarrow a \cdot b_1$
    for $i \leftarrow 2$ to n do
        if $b_i \leq 1$ then
            if $b_i = 1$ then                                       ▷ (and if $b_i = 0$ do nothing)
                $p \leftarrow$ ADD_WITH_OFFSET$(p, a, i)$                 ▷ $p \leftarrow p + a \cdot 2^{32 \cdot i}$
        else
            $p \leftarrow$ MUL_AND_ADD_WITH_OFFSET$(p, a, b_i, i)$        ▷ $p \leftarrow p + a \cdot b_i \cdot 2^{32 \cdot i}$
    return $p$
end procedure
```

for short ciphertexts

Further detail on the cryptanalysis



Algorithm 5 GnuPG's multiplication code (see function mpihelp_mul_karatsuba_case in mpi/mpih-mul.c).

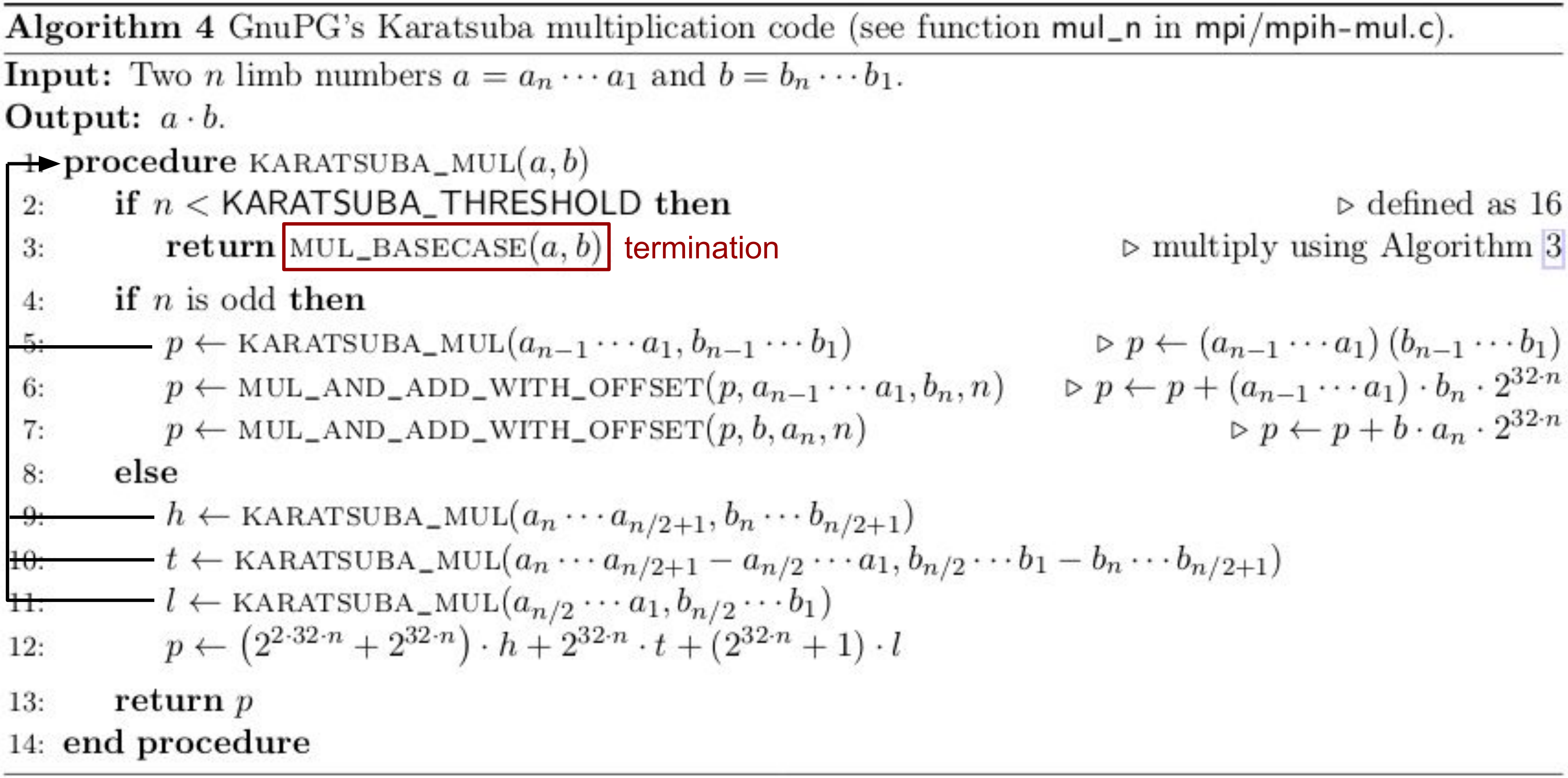
Input: Two numbers $a = a_k \cdots a_1$ and $b = b_n \cdots b_1$ of size k and n limbs respectively.

Output: $a \cdot b$.

```
procedure MUL($a, b$)    for long ciphertexts
    if $n <$ KARATSUBA_THRESHOLD then                          ▷ defined as 16
        return MUL_BASECASE($a, b$)                            ▷ multiply using Algorithm 3
    $p \leftarrow 0$
    $i \leftarrow 1$
    while $i \cdot n \leq k$ do
        $\boxed{t \leftarrow \text{KARATSUBA\_MUL}(a_{i \cdot n} \cdots a_{(i-1) \cdot n+1}, b)}$    ▷ multiply $n$ limb numbers using Algorithm 4
        $p \leftarrow$ ADD_WITH_OFFSET($p, t, (i-1) \cdot n$)    ▷ $p \leftarrow p + t \cdot 2^{32 \cdot (i-1) \cdot n}$
        $i \leftarrow i + 1$
    if $i \cdot n > k$ then
        $t \leftarrow$ MUL($b, a_k \cdots a_{(i-1) \cdot n+1}$)    ▷ multiply the remaining limbs of $a$ using a recursive call
        $p \leftarrow$ ADD_WITH_OFFSET($p, t, (i-1) \cdot n$)    ▷ $p \leftarrow p + t \cdot 2^{32 \cdot (i-1) \cdot n}$
    return $p$
end procedure
```

Further detail on the cryptanalysis

Karatsuba recursive algorithm is a very efficient way to perform **large integer multiplications**.



Algorithm 4 GnuPG's Karatsuba multiplication code (see function mul_n in mpi/mpih-mul.c).

Input: Two n limb numbers $a = a_n \cdots a_1$ and $b = b_n \cdots b_1$.

Output: $a \cdot b$.

```
procedure KARATSUBA_MUL($a, b$)
    if $n <$ KARATSUBA_THRESHOLD then                                    ▷ defined as 16
        return $\boxed{\text{MUL\_BASECASE}(a, b)}$ termination          ▷ multiply using Algorithm 3
    if $n$ is odd then
        $p \leftarrow$ KARATSUBA_MUL($a_{n-1} \cdots a_1, b_{n-1} \cdots b_1$)          ▷ $p \leftarrow (a_{n-1} \cdots a_1)(b_{n-1} \cdots b_1)$
        $p \leftarrow$ MUL_AND_ADD_WITH_OFFSET($p, a_{n-1} \cdots a_1, b_n, n$)      ▷ $p \leftarrow p + (a_{n-1} \cdots a_1) \cdot b_n \cdot 2^{32 \cdot n}$
        $p \leftarrow$ MUL_AND_ADD_WITH_OFFSET($p, b, a_n, n$)                      ▷ $p \leftarrow p + b \cdot a_n \cdot 2^{32 \cdot n}$
    else
        $h \leftarrow$ KARATSUBA_MUL($a_n \cdots a_{n/2+1}, b_n \cdots b_{n/2+1}$)
        $t \leftarrow$ KARATSUBA_MUL($a_n \cdots a_{n/2+1} - a_{n/2} \cdots a_1, b_{n/2} \cdots b_1 - b_n \cdots b_{n/2+1}$)
        $l \leftarrow$ KARATSUBA_MUL($a_{n/2} \cdots a_1, b_{n/2} \cdots b_1$)
        $p \leftarrow \left(2^{2 \cdot 32 \cdot n} + 2^{32 \cdot n}\right) \cdot h + 2^{32 \cdot n} \cdot t + (2^{32 \cdot n} + 1) \cdot l$
    return $p$
end procedure
```

Further detail on the cryptanalysis

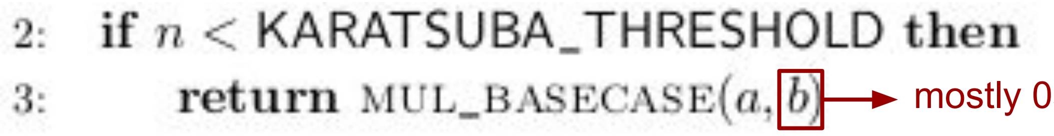
Each bit i in **q** could be:

- **q_i = 1**

In this case, following the multiplication routines to the Karatsuba algorithm:

The **second operand b** of the calls to **MUL_BASECASE** resulting **from** the **recursive calls** will contain **mostly zero limbs**.

KARATSUBA_MUL



```
if $n <$ KARATSUBA_THRESHOLD then
    return MUL_BASECASE($a$, $\boxed{b}$)
```

mostly 0s

Further detail on the cryptanalysis

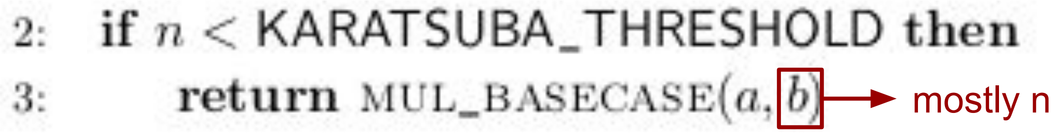
Each bit i in **q** could be:

- **$q_i = 0$**

In this case, following the multiplication routines to the Karatsuba algorithm:

The **second operand b** of the calls to **MUL_BASECASE** resulting **from** the **recursive calls** will contain **mostly (random-looking) non-zero limbs**.

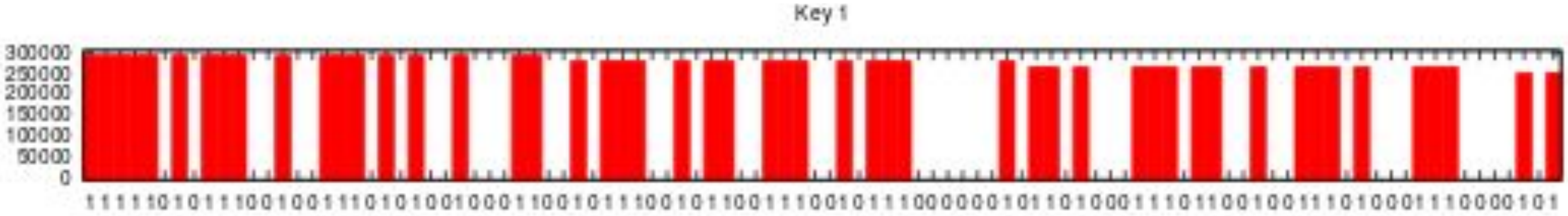
KARATSUBA_MUL



```
if $n <$ KARATSUBA_THRESHOLD then
    return MUL_BASECASE($a, \boxed{b}$)
```

mostly non-0s

Further detail on the cryptanalysis



Axis **X**: attacked **bit of q**

Axis **Y**: **#** of **zero limbs** in the **2° operand** of **MUL_BASECASE**

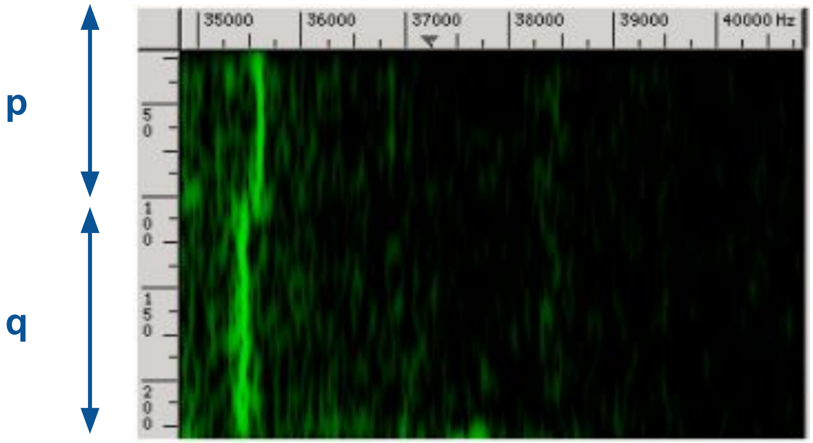
Large number of ***zero* limbs** → q_i = **1**

Low number of ***zero* limbs** → q_i = **0**

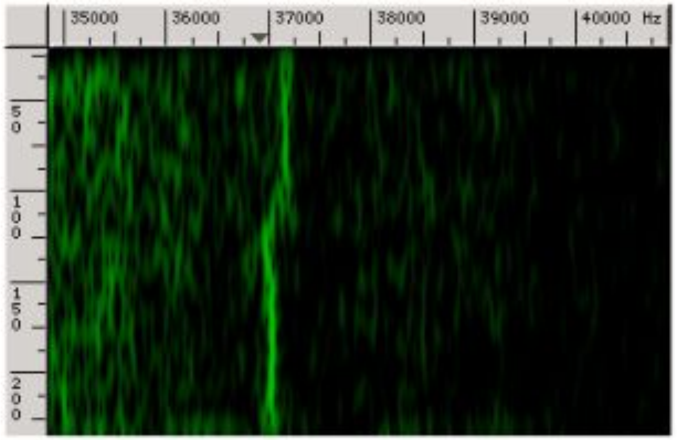
The **drastic change** in the number of **non-zero limbs** in the **second operand** of MUL_BASECASE is **detectable** by our **side channel measurements**.

Further detail on the cryptanalysis

Observation: generating **2 random ciphertexts** of respectively **63 limbs** and **57 limbs** (non-zero limbs):



63-limb ciphertext



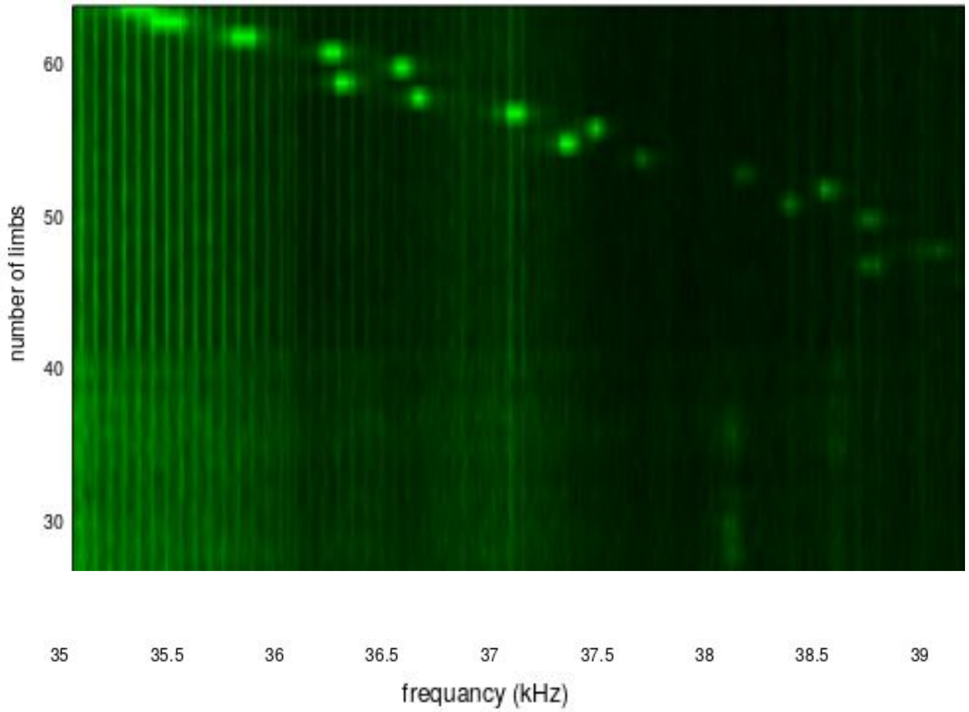
57-limb ciphertext

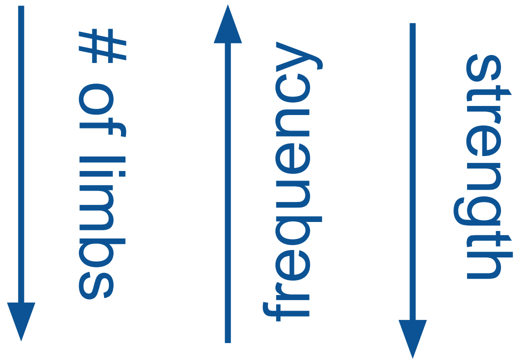
Decryption of the **63-limb** ciphertext produces a **signal** at **lower frequency** than the decryption of the **57-limb** ciphertext.

It can be ***found the num of limbs in the 2° operand of MUL.***

Further detail on the cryptanalysis

Therefore: the **shorter** the **number** of **limbs** (in 2° operand) the **higher** the **frequency** of the acoustic leakage, and the **weaker** the **signal strength**.




# of limbs

frequency

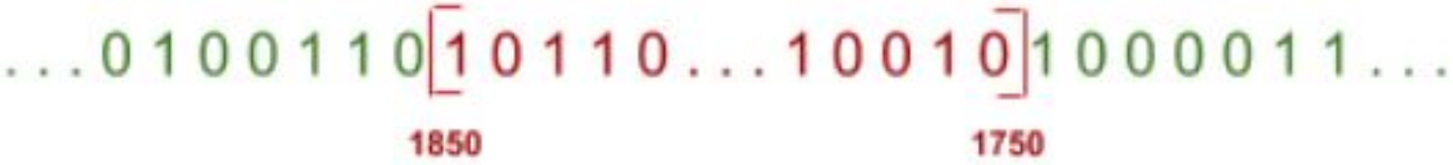
strength

We can **acoustically detect** when the 2° operand of MUL_BASECASE has **many non-zero limbs** (**$q_i = 0$**) or when it has **few non zero-limbs** (**$q_i = 1$**).

Problems

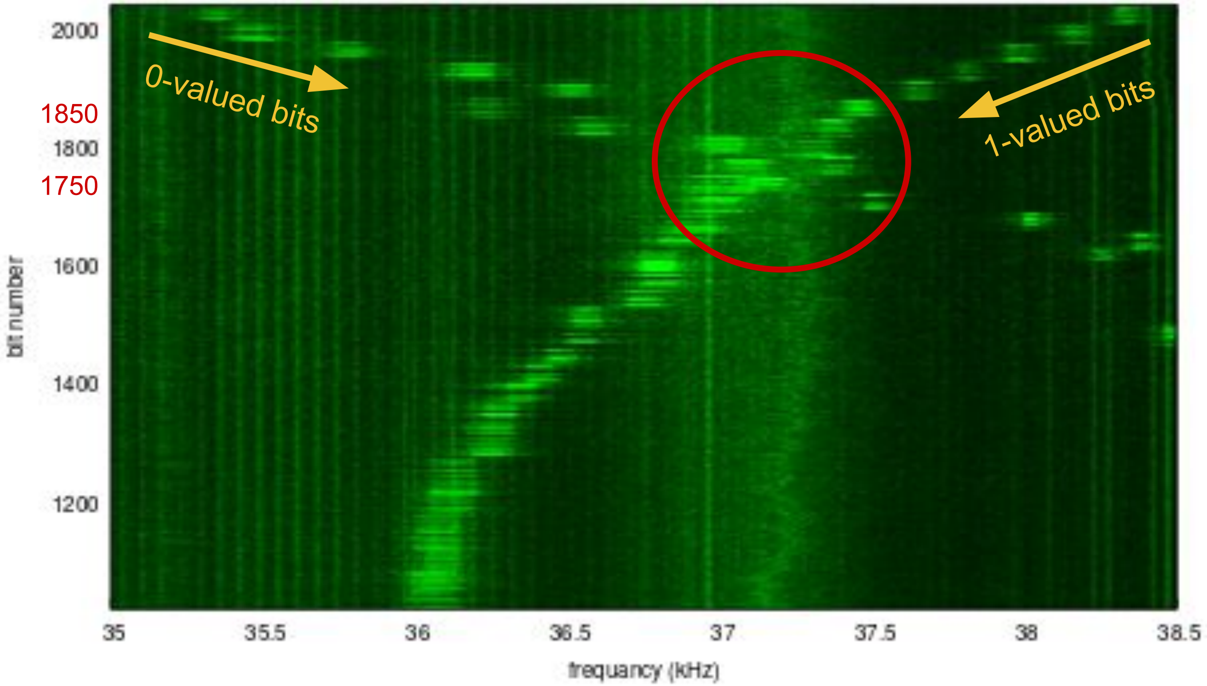
Unfortunately, there is a problem:

Distinguishing the **above two cases** using *side channel leakage* **is particularly hard for bits** in the rage of **1850–1750**.



This complication **requires us to use additional tricks**.

Problems

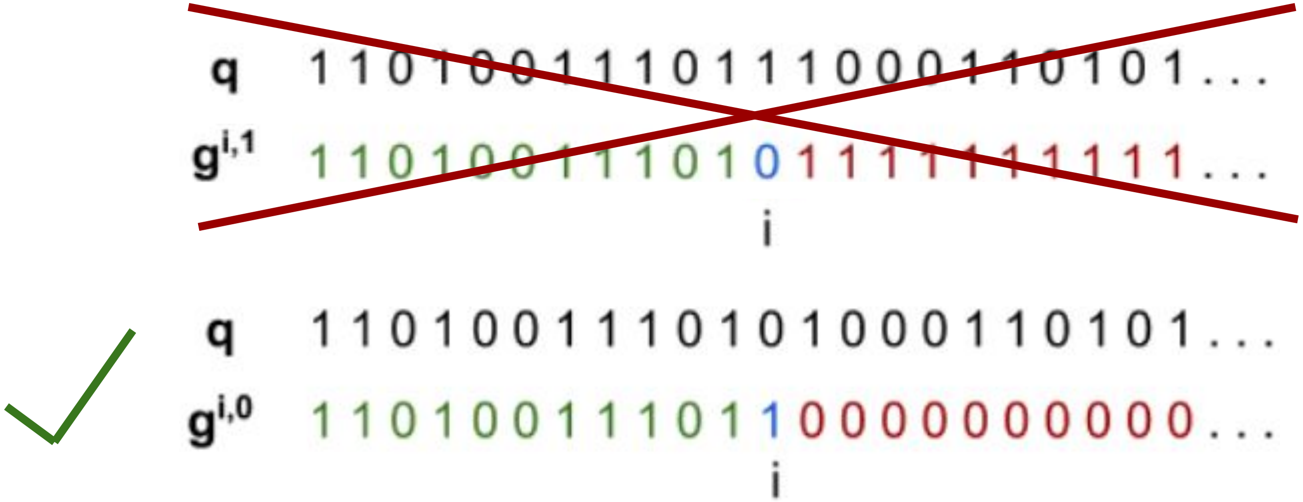


When it's used **c = g**^{i,1} **+ n**, bits in range [**1850 - 1750**] emit **very similar frequencies** with a distance of nearly 200Hz.

- The **bit index** where this **crossing point** occurs **depends on** the specific **values** of the **ciphertext used**!

Problems

Let **$g^{i,0}$** be **2048-bit number** whose **top i−1 bits** are the **same as q**, its **i-th bit is 1** and all **the rest** of its bits **are 0**.



~~q 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0 0 1 1 0 1 0 1 . . .~~

~~$g^{i,1}$ 1 1 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 . . .~~

~~i~~

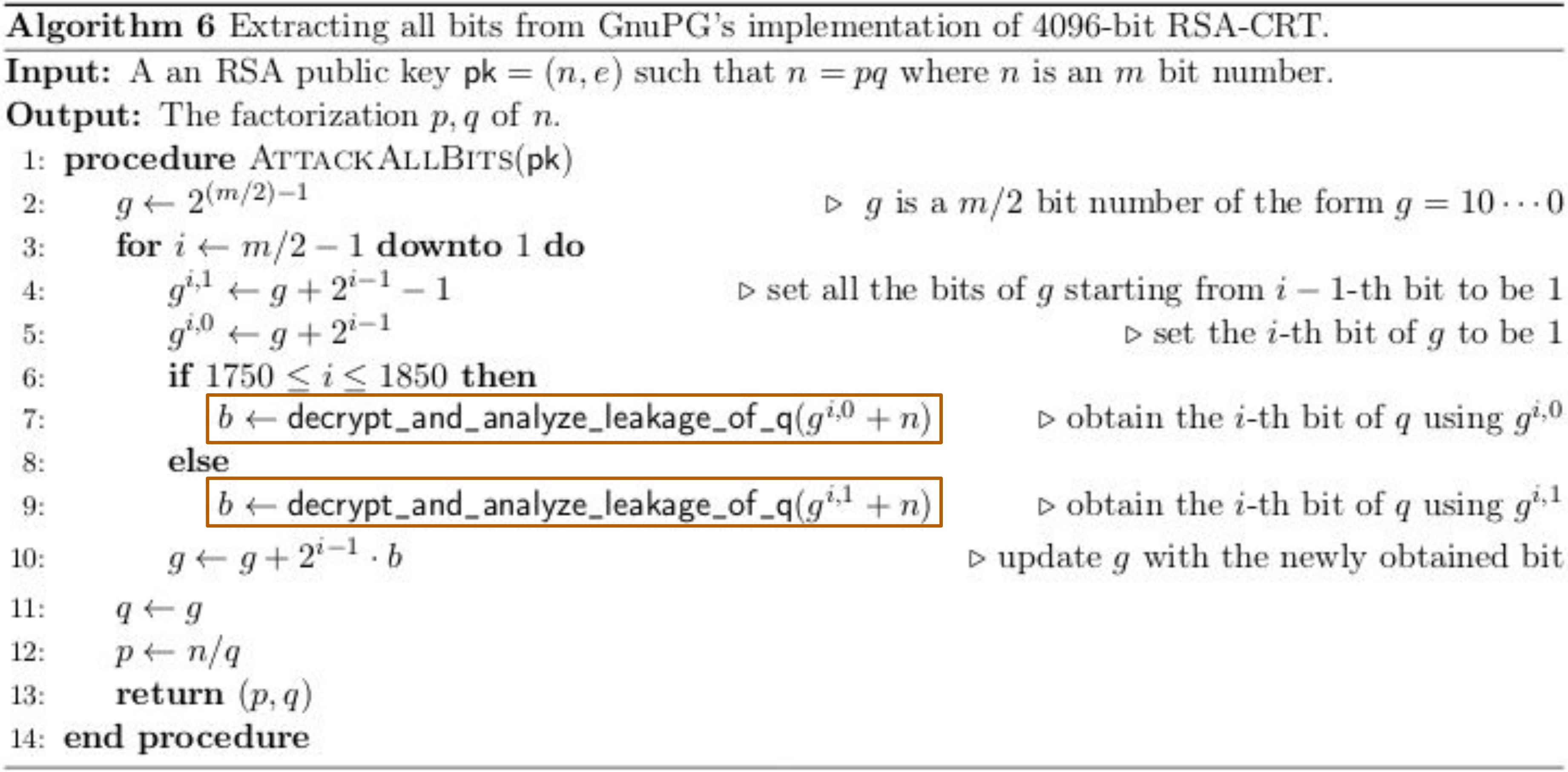
q 1 1 0 1 0 0 1 1 1 0 1 0 1 0 0 0 1 1 0 1 0 1 . . .

$g^{i,0}$ 1 1 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 . . .

i

Using **$g^{i,0}$** it is now **possible to distinguish** the **bits** in the range of **1750–1850** thus allowing our attack to proceed.

Problems



Algorithm 6 Extracting all bits from GnuPG's implementation of 4096-bit RSA-CRT.

Input: A an RSA public key $\mathsf{pk} = (n, e)$ such that $n = pq$ where n is an m bit number.

Output: The factorization p, q of n.

```
procedure AttackAllBits(pk)
    $g \leftarrow 2^{(m/2)-1}$                                   ▷ $g$ is a $m/2$ bit number of the form $g = 10\cdots0$
    for $i \leftarrow m/2 - 1$ downto 1 do
        $g^{i,1} \leftarrow g + 2^{i-1} - 1$                    ▷ set all the bits of $g$ starting from $i-1$-th bit to be 1
        $g^{i,0} \leftarrow g + 2^{i-1}$                        ▷ set the $i$-th bit of $g$ to be 1
        if $1750 \le i \le 1850$ then
            $\boxed{b \leftarrow \mathsf{decrypt\_and\_analyze\_leakage\_of\_q}(g^{i,0} + n)}$    ▷ obtain the $i$-th bit of $q$ using $g^{i,0}$
        else
            $\boxed{b \leftarrow \mathsf{decrypt\_and\_analyze\_leakage\_of\_q}(g^{i,1} + n)}$    ▷ obtain the $i$-th bit of $q$ using $g^{i,1}$
        $g \leftarrow g + 2^{i-1} \cdot b$                      ▷ update $g$ with the newly obtained bit
    $q \leftarrow g$
    $p \leftarrow n/q$
    return $(p, q)$
end procedure
```

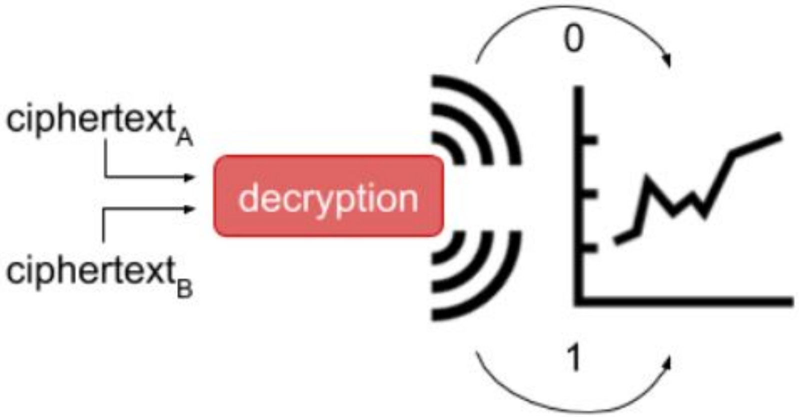
Problems

The attack proceeds in two stages:

1. Calibration stage

The attacker **generates two ciphertexts** corresponding to a **leakage of 0 and 1 bits** of q and **obtains multiple samples of** their **decryption**.

The attacker **generates a template of the leakage** caused by 0 bit and a template of the leakage caused by a 1 bit.



Problems

2. Attack stage

(2 steps)

- *Classification step*

A **spectrum of** an obtained **leakage** is **classified using** the **templates** as **corresponding** to **0 bit** or to a **1 bit**. This might be repeated a few times.

- *Template update step*

New templates for 0 bits and 1 bits are generated **updating** the **old ones** with the new leakages.

Error detection

If by mistake some bit **$q_j = 1$** is **misclassified as 0**, **successive values** of both **$g^{i,1}$** and **$g^{i,0}$** for all i < j will be **always <u>smaller</u> than q**.

This value will have the **same acoustic leakage** as if **q_i= 1**: next bits will result as **all 1s** regardless their actual value.

$$1\ 0\ 1\ 0\ 0\ \underset{0}{\cancel{1}}\ .\ .\ . \longrightarrow 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ .\ .\ .$$

- Solution: when a **sequence** (ex. 20 bits) of **only 1s** is detected, the attacker can **backtrack some bits** (ex. 50 bits) and **try again**.

Error detection

If by mistake some bit **q_j = 0** is **misclassified as 1**, **successive values** of both **$g^{i,1}$** and **$g^{i,0}$** for all $i < j$ will be **always <u>larger</u> than q**.

This value will have the **same acoustic leakage** as if **q_i= 0**: next bits will result as **all 1s** regardless their actual value.

1 0 1 0 0 $\underset{1}{\cancel{0}}$ . . . → **1 0 1 0 0 1 1 0 0 0 0 0 0 . . .**

- Solution: when a **sequence** (ex. 20 bits) of **only 0s** is detected, the attacker can **backtrack some bits** (ex. 50 bits) and **try again**.

Attack mitigation

Acoustic shielding: acoustic absorbers and sound-proof enclosures could **attenuate the signals**, but **do not prevent** the **attack**.

Noisy environment: **noise** in a noisy environment is **below 10 kHz**, acoustic **leakage** is well **above this rage**, such **noises** can be **filtered out**.

Parallel software load: perform the **computation** in **parallel** will **move** the **leakage frequency** from **35-38 kHz** to **32-35 kHz** (easier to detect).

Attack mitigation

Ciphertext randomization: instead of decrypting c, given a 4096-bit random value r, one can **decrypt** $\mathbf{r^e \cdot c}$ and **multiply** the **result by** $\mathbf{r^{-1}}$.

Ciphertext normalization: it can be **removed** all **leading zeros** of **c** and **decrypt c′** = c mod n. This value will have the **same limb** count **as q**, **line 2** of Alg1 will be **never taken**, making it **impossible** to use the modular reduction in order to **create a connection**.

Conclusion

- It’s possible with **some version of GPG RSA** (1.x) to attack a secret key with acoustic cryptanalysis.

- It’s **neither easy** nor **practical**.

- To carry out this kind of attack is required **time** and **effort**.

- It could be **mitigated**.

BUT IT IS POSSIBLE WITH A SMARTPHONE TO FIND AN RSA SECRET KEY!

References

[1] Daniel Genkin, Adi Shamir and Eran Tromer, RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. CRYPTO 2014

[2] Adi Shamir, Eran Tromer, Acoustic Cryptanalysis - On nosy people and noisy machines