A common weakness in RSA signatures: extracting public keys from communications and embedded devices

Hackito Ergo Sum
24-26 April 2014

Renaud Lifchitz
renaud.lifchitz @ oppida.fr
Speaker’s bio

- French computer security engineer working at Oppida, France

- Main activities:
  - Penetration testing & security audits
  - Security research
  - Security trainings

- Main interests:
  - Security of protocols (authentication, cryptography, information leakage…)
  - Number theory (integer factorization, primality testing…)

Hackito Ergo Sum 2014 – 24-26 April

“A common weakness in RSA signatures: extracting public keys from communications and embedded devices”, Renaud Lifchitz
RSA signature basics
Asymmetric cryptography is widely used to do digital signatures:
- Private keys are used to digitally sign messages
- Corresponding public keys are used to verify signatures
- Integer factorization allows an attacker to find the private keys from public ones, but is generally hard

Public keys are almost always transmitted out-of-band (public key server, local keystore) before communication/usage

One of the most used signature scheme is RSA signature
Introduction – RSA signature

• Steps to sign a message using RSA:

  – Message $m$ is hashed using a hash algorithm $h(\cdot)$: MD5, SHA1, SHA256, …
  – Hash is then padded to avoid forgery by multiplication, using a padding algorithm $p(\cdot)$ like PKCS
  – The result is raised to the $d$-th power and reduced modulo $n$, where $d$ is the private exponent and $n$ is the public key

$$p(h(m))^d \equiv s \pmod{n}$$
Extracting public keys from signed messages
The idea

- Suppose we have 2 different messages with their corresponding signatures \((m_1, s_1), (m_2, s_2)\) with unknown public key \(n\):

\[
\begin{align*}
    p(h(m_1))^d &\equiv s_1 \pmod{n} \\
    p(h(m_2))^d &\equiv s_2 \pmod{n}
\end{align*}
\]

\[
\Rightarrow \quad \begin{cases} 
    p(h(m_1)) \equiv s_1^e \pmod{n} \quad \text{with quotient } q_1 \\
    p(h(m_2)) \equiv s_2^e \pmod{n} \quad \text{with quotient } q_2
\end{cases}
\]

by Euler theorem

\[
\Rightarrow \quad \gcd(s_1^e - p(h(m_1)), s_2^e - p(h(m_2))) = \gcd(q_1, q_2).n
\]

which gives a small (probably smooth) multiple of public key \(n\)
The idea

• Then we have to remove all small factors from the result until the residue size is a well-known asymmetric key size (512, 768, 1024, 2048, 4096 bits…)

• Trial division is sufficient in 99.9999% of cases, otherwise we can use an additional signed message in the GCD or use ECM factoring algorithm to help

• We now have computationally extracted our unknown public key!
Requirements

- Hash and padding algorithms must be known or guessed
- $e$ should be small because computation will be done without modular arithmetic
- $n$ should be small to medium
• Main limitation is memory consumption

• The computation:
  – takes about $O(e \log(n))$ bits of memory
  – costs about:
    • $O(\log(e))$ big integer multiplications (exponentiation step)
    • $O(e \log(n))$ big integer divisions (GCD step)
Applications

- Without access to any kind of keyserver nor keystore and being entirely passive, we can:
  - Extract public keys used in RSA signatures
  - Authenticate subsequent messages
  - Find people or devices using weak keys that weren’t discoverable before: **this gives a new angle of attack for embedded devices/blackbox protocols using RSA signatures**
  - Safely test whether different messages are signed using the same key/come from the same person (without relying on any kind of spoofable key id)
State of the art of factorization algorithms
Introduction

- There exists several algorithms for integer factorization, more or less naive

- Some algorithms are generic and can factor any number, some are form-specific

- Key generation weaknesses:
  - $p$ and $q$ too close
  - $p-1$, $q-1$, $p+1$ and/or $q+1$ too smooth
  - weak RNG (Random Number Generator)

- A generic but good open source program for factoring:
  Yafu (http://sourceforge.net/projects/yafu/)
Finding small factors in large integers

- **Trial factoring:**
  when there are very small factors (less than 10 digits)
- **Pollard Rho:**
  for small factors
- **Pollard’s P-1:**
  when one or more factors are p-1 smooth
- **Williams’ P+1:**
  when one or more factors are p+1 smooth
- **Elliptic Curve Method (ECM):**
  for factors up to 80 digits
Finding large factors in small integers

- **Fermat algorithm:**
  when a factor and its co-factor are really near in absolute value

- **Quadratic sieve (QS):**
  faster and simpler NFS for integers < 100 digits

- **Number Field Sieve (NFS):**
  for integers of intermediate size

- **General Number Field Sieve (GNFS):**
  for numbers up to 230 digits (RSA-768)

- **Special Number Field Sieve (SNFS):**
  for numbers with specific form ($r^e \pm s$ with $r$ and $s$ small)
  up to 320 digits
Practical applications - PGP
What is PGP?

- Pretty Good Privacy (PGP) is a data encryption and decryption program mostly used for securing e-mails
- Created in 1991 by Phil Zimmermann
- Software: PGP (Windows) / GnuPG (Linux)
- OpenPGP standard (RFC 4880)
Computation steps to extract public key - PGP

- Prepare original message before hashing:
  - Canonicalize message (newlines are converted to \r\n)
  - Append specific PGP data:
    - PGP version
    - Signature type
    - Public algorithm (here RSA)
    - Hash algorithm
    - Signature date & time
  - Recreate PKCS#1 padded ASN.1 message hash following RFC 4880
- Compute:
  \[ \gcd(s_1^{65537} - p(h(m_1')), s_2^{65537} - p(h(m_2'))) \]
Just a proof-of-concept:

- Supports RSA signature with SHA-1 hashing only
- Not optimized (mixed Python + PARI-GP implementation, would be faster in C)

Able to find the signing public key of anybody using only 2 signed mails!
« A common weakness in RSA signatures: extracting public keys from communications and embedded devices », Renaud Lifchitz
Practical applications - Vigik access control system
What is Vigik?

- French access control for residential buildings (nearly 1 million buildings are protected by Vigik in France)
- Contactless system
- Made to replace the old T25 lock and avoid existing master keys
- 2 kinds of tokens:
  - Resident tokens (various contactless protocols, not interesting), can access a given building at any time
  - Service tokens (based on Mifare Classic + RSA signature of 768 or 1024 bits), can access all buildings during specific time slots
- May be used for other kinds of access control like ATMs or military premises
What is Vigik?

Vigik contactless reader

Resident token

Service token
What is Vigik?

- 4 common types of service tokens:
  - « La Poste Service Universel »:
    service code 0x7AA, authorized access from Monday to Saturday, 6:00-0:00 (may vary)
  - « La Poste Autre Services »:
    service code 0x7AB, authorized access any day, 6:00-0:00 (may vary)
  - « France Telecom »:
    service code 0x7AC, authorized access any day, any time
  - « EDF-GDF »:
    service code 0x7AD, authorized access any day, any time
What is Vigik?

- Service tokens need to be loaded with a valid RSA signature for the current date & time slot

- For instance, the postmen load their token every morning before mail delivery

- A token can be loaded in advance but for no more than 3 slots of 84 successive hours for security reasons (to mitigate token loss or theft risks)
Vigik storage

- Vigik uses NXP Mifare Classic 1K cards as storage
- 16 sectors of 4 blocks = 64 blocks of 16 bytes
- Last block of each sector is reserved for A and B keys and ACL
- RSA signatures are splitted across several blocks/sectors
Attacks against protocol
Reader-only attacks

According to NXP, manufacturer of Mifare products, Vigik has registered prefix code 0x49 (see NXP AN10787 document).

With no valid service token, using a blank Mifare Classic card, and by crafting several MAD (Mifare Application Directory) structures, sniffing using a Proxmark3 RFID device, we have noticed that a 0x4910 entry triggers a sector 1 read.

Sector authentication in Mifare Classic is badly designed: reader authenticates itself first. It is possible for the card to send many challenges and gather all the answers for an offline cracking.

Using a crapto1 library, it becomes possible to crack the 48-bit sector access key A which happens to be:

0x314B49474956

("1KIGIV" in ASCII, to be read in reverse order...)

Hackito Ergo Sum 2014 – 24-26 April
« A common weakness in RSA signatures: extracting public keys from communications and embedded devices », Renaud Lifchitz
Attacks against protocol
Card-only attacks

- With no valid Vigik reader, it is possible to retrieve all sector keys using the well-known offline nested attack by Nethemba (mfoc tool), as sector 0 key A is default key 0xA5A4A3A2A1A0

- We find that key A for other sectors is « 1KIGIV » and that key B is proprietary and can vary between cards
Attacks against protocol

• With the knowledge of A and B keys, we are now able to:
  – Dump and analyze any service token
  – Clone any service token
    (for instance using a Chinese programmable UID Mifare Card)
  – Emulate any service token
    (for instance using a Proxmark3)
Vigik card layout reverse-engineered

Hackito Ergo Sum 2014 – 24-26 April

« A common weakness in RSA signatures: extracting public keys from communications and embedded devices », Renaud Lifchitz
Hardware attack: dumping the reader flash memory

Dumping the flash memory of a Vigik reader using a Teensy 2
(thanks to Gric for his help!)

Hackito Ergo Sum 2014 – 24-26 April
« A common weakness in RSA signatures: extracting public keys from communications and embedded devices », Renaud Lifchitz
Hardware attack: dumping the reader flash memory

- Extracted 1024-bit public keys (1/2):
  - « La Poste Service Universel »:
    0xAB9953CBFCCD9375B6C028ADBAB7584BED15B9CA037FADED976599
    6F9EA1AB983F3041C90DA3A198804FF90D5D872A96A4988F91F2243B
    821E01C5021E3ED4E1BA83B7CFECAB0E766D8563164DE0B2412AE4E6
    EA63804DF5C19C7AA78DC14F608294D732D7C8C67A88C6F84C0F2E3F
    AFAE34084349E11AB5953AC68729D07715
  - « La Poste Autres Services »:
    0xA6D99B8D902893B04F3F8DE56CB6BF24338FEE897C1BCE6DFD4EBD
    05B7B1A07FD2EB564BB4F7D35DBFE0A429666C2C137AD156E3DAB6290
    4592BCA20C0BC7B8B1E261EF82D53F52D203843566305A49A22062DE
    CC38C2FE3864CAD08E79219487651E2F79F1C9392B48CAFE1BFFAFFE4
    802AE451E7A283E55A4026AD1E82DF1A15
Hardware attack: dumping the reader flash memory

Extracted 1024-bit public keys (2/2):

- « France Telecom »:
  0xC44DBCD92F9DCF42F4902A87335DBB35D2FF530CDB09814CFA1F4B95A1BD018D099BC6AB69F667B4922AE1ED826E72951AA3E0EAAA7D49A695F04F8CDAAE2D18D10D25BD529CBB05ABF070DC7C041EC35C2BA7F58CC4C349983CC6E11A5CBE828FB8ECBC26F08E1094A6B44C8953C8E1BAFD214DF3E69F430A98CCC75C03669D

- « EDF-GDF »:
  0xB35193DBD2F88A21CDCF4F4BF84F7FC036A991A363DCB3E802407A5E5879DC2127EECF520779E79E911394882482C87D09A88B0711CBC2973B77FFDAE40EA0001F595072708C55B484AB89D02BCB971FF1B80371C0BE30CB13661078078BB68EBCCA524B9DD55EBF7D47D9355AFC95511350CC1103A5DEE847868848B235
Vigik RSA signature

- I have discovered that Vigik uses deprecated ISO 9796-2 for RSA signature with:
  - Public key $N \equiv 5 \pmod{8}$
  - $p$ and $q$ of the form $8k+3$ and $8k+7$ (without order)
  - Public exponent for speed purposes is $e = 2$ (even)
    and $e \cdot d \equiv 1 \pmod{\frac{(p-1)(q-1)}{4}}$
  - It implies $d = \frac{(p-1)(q-1)+4}{8}$

- It follows that Vigik is vulnerable to some attacks described in:
  but chosen-plaintext attacks are not possible in this case
Vigik security in the next few years

• Interestingly, RSA key for « La Poste Service Universel » has already been changed (key version = 2 in the dump), has the key been compromised?

• Token storage (Mifare Classic) is broken since several years now

• Token signature is within range of direct factoring attacks because weak public keys can be extracted:
  – RSA 768 is already broken (December 2009)
  – RSA 1024 will probably be publicly broken by researchers within 3-4 years
• Because of hardware and storage constraints, key sizes in Vigik are not upgradable (maximum keysize is 1024 bits)

• Full service token forgery will happen in the next few years

• Vigik system is to be changed

• Replacement of 1 million Vigik readers will cost several hundred million euros, upgradable security would have saved this cost
Countermeasures
• The « problem » comes from deterministic padding. RSA encryption uses random padding to avoid various attacks. This is not the case in RSA signature. It would be possible to use non-deterministic padding in signature to avoid public key leaks (like RSA-PSS scheme).

• Other signatures schemes may or may not be vulnerable to this attack (this exercise is left to the reader!)

• In all cases, use strong keys and large enough public exponents.
Thanks!

Any questions?