Katacryptology

Robert Erra
LSE WEEK 2014

July 17, 2014

Plan

1. Catacrypt ?

Catacrypt ? Chairman: J.-J. Quisquater

From http://catacrypt.net/ :

The main point is: many cryptographic protocols are only based on the security of one cryptographic algorithm (e.g. RSA) and we don't know the exact RSA security (including Ron Rivest). What if somebody finds a clever and fast factoring algorithm? Well, it is indeed an hypothesis but we know several instances of possible progress. A new fast algorithm is a possible catastroph if not handled properly. And there are other problems with hash functions, elliptic curves, aso. Think also about the recent Heartbleed bug (April 2014, see http://en.wikipedia.org/wiki/Heartbleed): the discovery was very late and we were close to a catastrophic situation.

Plan

❷ Why should you trust or not your favorite cryptotool ?

Rise and Fall of a cryptographic tool: 1

There is a *Pre and Post Snowden Era*:

Pre Snowden Era Problems:

- PreSE 1: Your key got older
- PreSE 2: Your algorithm got older
- PreSE 3: Your(s) key(s) has(have) been badly computed
- PreSE 4: Your algorithm has been badly programmed[1]

[1] See https://cryptocoding.net

Rise and Fall of a cryptographic tool: 2

Pre Snowden Era Problems:

- PreSE 5: Your algorithm is used into an insecure protocol
- PreSE 6: Your software is used on an insecure device (a smart card)
- PreSE 7: Your processor has a bug (we can use the intel division bug to factor a RSA key)

Rise and Fall of a cryptographic tool: 3

Post Snowden Era Problems:

- PostSE 1: Your algorithm has a backdoor
- PostSE 2: Your processor has a backdoor
- PostSE 3: Your key has a backdoor
- and so on . . .

Rise and Fall of a cryptographic tool: 4

To trust your favorite cryptotool means:

You have to trust the *full stack*:

- To trust your algorithm
- To trust your code
- To trust all the computations that use *randomness*
- To trust the protocols that uses your cryptotool
- To trust your processor
- To trust your device
- and so on . . .

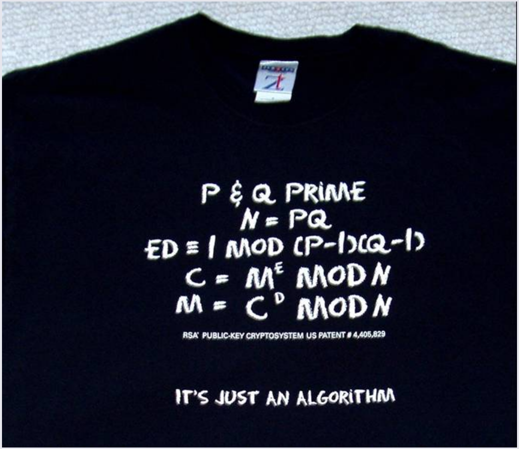
Remind: Attackers can do what they want! So, is it possible to trust your favorite cryptotool?

Plan

3 RSA

RSA: A "joke"

RSA on the shirt/gift september 17th 2000 (end of the patent RSA (1978)) par *RSA Data Securiy*!



RSA

RSA: Rivest Shamir Adleman (cf wikipedia)

- RSA is an algorithm for public-key cryptography . . .
- . . . publicly described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT
- . . . and probably invented by Clifford Cocks, a British mathematician working for the UK intelligence agency GCHQ
- . . . and is believed to be secure given sufficiently long keys and the use of up-to-date implementations.

RSA: Computing the keys

1. Choose two ≠ large random prime numbers p and q.
2. Compute $n = p \times q$; n is the modulus
3. Compute the Euler totient: $\varphi(n) = (p-1)(q-1)$.
4. Choose an integer e such that $1 < e < \varphi(n)$, and e is coprime to $\varphi(n)$: $gcd(e, \varphi(n)) = 1$.
5. Compute d to satisfy the congruence equation $d\,e \equiv 1 \bmod \varphi(n)$, i.e. for some integer k:

$$d\,e = 1 + k\varphi(n)$$

RSA: vocabulary

Remarks

- e is released as the public key exponent.
- d is kept as the private key exponent
- The public key consists of the modulus n and the public (or encryption) exponent e: (n, e).
- The private key consists of the modulus n and the private (or decryption) exponent d which must be kept secret: (n, d).

RSA: how to use it ?

Suppose Alice uses Bob's public key to send him

. . . an encrypted message. But Bob has no way of verifying that the message was actually from Alice since anyone can use Bob's public key to send him encrypted messages. So, in order to verify the origin of a message, RSA can also be used to sign a message.

- Encryption: $C = M^e \bmod n$
- Decryption: $M = C^d \bmod n$
- Signature: $S = M^d \bmod n$

Plan

4 PreSE 1

PreSE 1: factoring an RSA modulus ?

The Humpich affair

- In 1997 S. Humpich factored the GIE CB Public RSA modulus
- It has been chosen years before: 320 bits
- In 1991 the record was RSA100: which means 330 bits
- In 2000 the record was RSA512: which means 512 bits
- S. Humpich was put in jail . . .

Factorization

— Some factorisation records

N	Year	Algorithm
RSA-120 (399 bits)	1993	MQPS
RSA-129 (429 bits)	1994	MPQS
RSA-130 (432 bits)	1996	NFS
RSA-140 (466 bits)	1999	NFS
RSA-155 (512 bits)	1999	NFS
RSA-160 (532 bits)	2003	NFS
RSA-200 (665 bits)	2005	NFS
RSA-768 bits	2010	NFS
RSA-1024 bits	$2030^{?}$	??

Plan

5. PreSE 2

PreSE 2: Some examples of Algorithms

Dead or Alive ?

Name	Size (bits)	Word size	Alive ?
MD2	128	32	Dead
MD4	128	32	Dead
DES*	64	64	Dead
MD5	128	32	Dead
RC4 (WEP)	64	64	Dead
SHA-1	160	40	At Death's door
SHA-256/224	256/224	32	Alive
SHA-512/384	512/384	64	Alive
TIGER-160/192	160/192	64	Alive

Plan

6 PreSE 3

PreSE 3: Computing secure keys

The Debian/OpenSSL case (2008) (ref. 7)

- In order to keep a warning from being issued by the Valgrind analysis tool, a maintainer of the Debian distribution applied a patch to the Debian implementation of the OpenSSL suite, which inadvertently broke its random number generator in the process.
- Any key generated with the broken random number generator, as well as data encrypted with such a key, was compromised.
- The error was reported by Debian on May 13, 2008.
- This bug resulted in very weak keys being generated for SSH, SSL Certificates, OpenVPN and other uses.

Plan

7 PreSE 5

PreSE 5: Insecure protocol (ref. 8)

- Adaptive-chosen-ciphertext attacks were largely considered to be a theoretical concern until 1998, when Daniel Bleichenbacher of Bell Laboratories demonstrated a practical attack against systems using RSA encryption in concert with the PKCS#1 v1 encoding function, including a version of the Secure Socket Layer (SSL) protocol used by thousands of web servers at the time.
- The Bleichenbacher attacks, also known as the million message attack, took advantage of flaws within the PKCS #1 function to gradually reveal the content of an RSA encrypted message.

PreSE 5: Insecure protocol (ref. 8)

So ?

Doing this requires sending several million test ciphertexts to the decryption device (e.g., SSL-equipped web server.) In practical terms, this means that an SSL session key can be exposed in a reasonable amount of time, perhaps a day or less.

Plan

8 PostSE 1

PostSE 1: Dual_EC_DRBG is . . . (ref. 9)

- Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG)[1] is a claimed cryptographically secure pseudorandom number generator (CSPRNG)
- Standardized by ANSI, ISO, and formerly by the National Institute of Standards and Technology (NIST).
- Dual_EC_DRBG is based on the elliptic curve discrete logarithm problem (ECDLP) and was for some time one of the four (now three) CSPRNGs standardized in NIST SP 800-90A.

PostSE 1: . . . backdoored (probably by NSA)

2014

Following a public comment period and review, NIST removed Dual_EC_DRBG as a cryptographic algorithm from its draft guidance on random number generators, recommending "that current users of Dual_EC_DRBG transition to one of the three remaining approved algorithms as quickly as possible."

Plan

9 PostSE 3

PostSE 3: Backdoored RSA Keys (ref. 10)

- The idea is: you have bought an RSA key from a company
- They have computed p, q, d, e
- such that n is easy to factor
- or d is easy to find
- Naive Example: pick p and $2p + 1$ primes and just put $n = p(2p + 1)$
- and Solve[X (2 X + 1) == n, X] gives you p!
- With $p = 10^{100} + 9223$ it takes 0.015s with Mathematica to factor by Solve.

Plan

10 Back to RSA . . . weaknesses

Back to RSA

— RSA is without doubt the most famous asymmetric cryptosystem.

In the building 10 of MIT, one can read . . .

Ronald Rivest, Adi Shamir and Leonard Adelman invented the first workable public key cryptographic system, based on the use of very large prime numbers, that has so far been proved unbreakable.

But:

- Technically, the last sentence is unfortunately *false*!
- There is no proof of "unbreakability" of RSA.

PreSE 5

— Attacks against the Public Exponent e :

- Cycling attack (Norris & Simmons)
  1. given $C = M^e \bmod N$
  2. compute $C_i = C^{e^i} \bmod N$ until we find $C_r = C^{e^r} = C \bmod N$
  3. then $M = C^{e^{r-1}} \bmod N$
- Common modulus attack (Simmons): we can find M with the hepl of the (famous) Bézout Indentity if:
  1. we are given $\{C_1 = M^{e_1} \bmod N, C_2 = M^{e_2} \bmod N\}$
  2. $\gcd(e_1, e_2) = 1 : e_1 u + e_2 v = 1$, so $M = \{C_1^u * C_2^v \bmod N\}$
- Broadcast attack (Hastad): we can find M if
  1. we are given $\{C_i = M^e \bmod N_i\}_{i=1}^{f}$
  2. $M^f < N_1 N_2 \cdots N_f$
- Small public exponent attack (based on Lattices Attacks)

National Agencies: what they say

— Some Recommendations from some National Agencies

- NIST (NIST-SP-800-89) in 2006: quite nothing about e, it is only recommanded that e has to be odd.
- DCSSI[a] in 2006: recommands to use public exponent strictly superior to $2^{16} = 65536$.
- FIPS in june 2009: it is proposed to select e *prior* to generating the primes p and q, and that the exponent e **shall**[b] be an odd positive integer such that:

$$2^{16} < e < 2^{256}. \tag{1}$$

[a] Now *ANSSI*.

[b] From FIPS: *Shall : Used to indicate a requirement of this Standard*.

GnuPG v1.2.3

— Algorithm used by GnuPG v1.2.3 to compute e *after* the computation of p and q.

```
Algorithm 1 :   Computation of $e$
    . . .
    If $\varphi(N) \neq 0 \bmod [41]$ Then $e = 41$;
    Else If $\varphi(N) \neq 0 \bmod [257]$ Then $e = 257$;
    Else
        $e = 65537$;
        While $GCD(e, \varphi(N)) \neq 1 : e = e + 2$;
```

GnuPG v1.2.3

— Analysis of GnuPG v1.2.3

Nguyen (1999) has shown this algorithm creates a minor flaw:

- if $e \geq 65539$ then $\varphi(N) = 0 \bmod [41 * 257 * 65537]$,
- and since $\varphi(N) = 0 \bmod [4]$
- we obtain a 32 bits factor of $\varphi(N)$!

This is not a serious threat because the probability to have $e \geq 65539$ is low ($< 0.2\%$) and the obtained knowledge of 32 bits of $\varphi(N)$ is not enough to be used in known efficient factorization algorithm. But, this can be useful for example

- to improve the Wiener attacks
- to improve some partial key exposure attacks.

GnuPG v1.4.10

— RSA in GnuPG v1.4.10: $e \geq 65537$

```
**Algorithm** 2 :    RSA key generation
    **Input**: — an integer k > 0;
    **Output**: — $(N, e, d)$ with $N$ a $k$ bit number
    **Begin**:
        $e = 65537$;
        **While** $bitSize(N) \neq k$
            Compute randomly a prime $p$ of $k/2$ bits;
            Compute randomly a prime $q$ of $k/2$ bits;
            Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;
        **While** $GCD(e, \varphi(N)) \neq 1$ $e = e + 2$;
        /* Again: if $e > 65537$, we gain information about $\varphi$(N) */
        Compute $d = e^{-1} \bmod \varphi(N)$ ;
    **End.**
```

Libgcrypt 1.4.4

— RSA in libgcrypt 1.4.4: $e \geq 65537$

```
**Algorithm** 3 :   RSA key generation (it follows ANS X9.31)
    **Input**: — an integer k = 1024 + 256s > 0;
    **Output**: — $(N, e, d)$ with $N$ a $k$ bit number
    **Begin**:
        $e = 65537$;
        Compute randomly a prime $p$ of $k/2$ bits;
        Compute randomly a prime $q$ of $k/2$ bits;
        Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;
        Compute $\lambda(N) = lcm(p-1, q-1) = \varphi(N)/gcd(p-1, q-1)$
        **While** $GCD(e, \lambda(N)) \neq 1$ $e = e + 2$;
        /* Again: if $e > 65537$, we gain information about $\lambda(N)$ */
        Compute $d = e^{-1} \bmod f$ ;
    **End.**
```

Can we do better ? Yes . . .

— RSA in OpenSSL 0.9.8k

```
Algorithm 4 :   RSA key generation
    Input: — an integer k ;
    Output: — $(N, e, d, d_p, d_q)$ with $N$ a $k$ bit number
    Begin:
        $e = 65537$; /* $e$ is fixed, $p$ and $q$ are recomputable */
        While $gcd(e, p-1) \neq 1$ Compute rand. prime $p$ of $k/2$ bits;
        While $gcd(e, q-1) \neq 1$ Compute rand. prime $q$ of $k/2$ bits;
        Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;
        Compute $d = e^{-1} \bmod \varphi(N)$ ;
        Compute $d_p = d \bmod (p-1)$ ;
        Compute $d_q = d \bmod (q-1)$ ;
    End.
```

How to compute d ?

— What about the private exponent d ?

The private exponent d is computed generally after e (duality). The main point is to avoid small d.

- Wiener attack (1990): $d < 1/4N^{1/3}$ is a *sufficient* condition to find the private exponent d in a time in the order of $O(\log N)$.
- Boneh and Durfee (2000): if $d < N^{0.292}$ then we can recover it with the help of the Coppersmith's method to solve modular equations (heuristic but it works!).
- It is recommanded by DSSCI to use private exponents of the same length as the RSA modulus.
- Conjecture (Boneh and Durfee): if $d < \sqrt{N}$ then RSA is insecure.

Plan

11 Quantum computers and the future of RSA ?

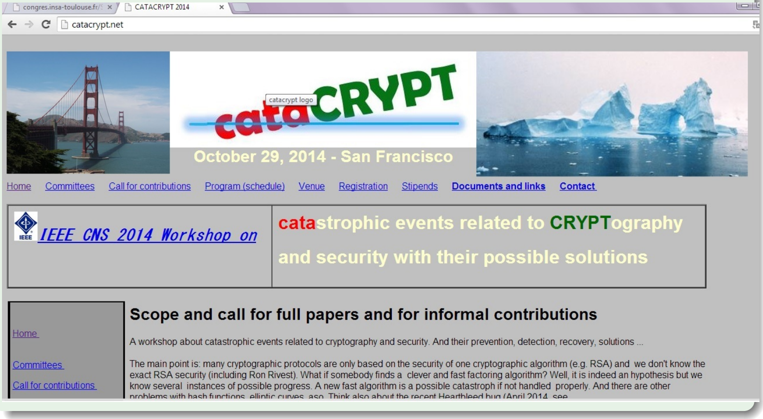
Quantum computers and the future of RSA

- A 2048 qubits quantum computer can factor very easily a RSA modulus of more (probably) than 1500 (classical) bits.
- Such a quantum computer will kill quite all classical asymmetric cryptographic algorithms based on RSA and DL.
- And we still wait for a quantum algorithm that could encrypt and would resist to a . . . quantum computer!
- But it can not factor easily an AES-256 bits key. It has been proved that by a brute force attack it will take $O(2^{128})$ to do it.

A (non classical) conclusion:

You can not trust your full stack! But there is hope . . .

Catacrypt: A workshop about catastrophic events related to cryptography and security. And their prevention, detection, recovery, solutions ...



Scope and call for full papers and for informal contributions

A workshop about catastrophic events related to cryptography and security. And their prevention, detection, recovery, solutions ...

The main point is: many cryptographic protocols are only based on the security of one cryptographic algorithm (e.g. RSA) and we don't know the exact RSA security (including Ron Rivest). What if somebody finds a clever and fast factoring algorithm? Well, it is indeed an hypothesis but we know several instances of possible progress. A new fast algorithm is a possible catastroph if not handled properly. And there are other problems with hash functions, elliptic curves, aso. Think also about the recent Heartbleed bug (April 2014, see

Another (non classical) conclusion:

. . . because, of course, you will still use your favorite cryptotool!

B. Schneier

Encryptions works. Properly implemented strong cryptosystems are one of the few things that you can rely on.

And remember this famous citation (?)

If it bleeds, we can kill it!

Some lectures:

1. MISC HS numéro 5: Avril / Mai 2012
2. MISC HS numéro 6: Novembre / Décembre 2012
3. MISC numéro 65: Janvier/Février 2014
4. Fluhrer, S. Mantin, I. and Shamir A. - Weaknesses in the Key Scheduling Algorithm of RC4.
5. http://fr.wikipedia.org/wiki/Serge_Humpich
6. https://security-tracker.debian.org/tracker/CVE-2008-0166
7. http://en.wikipedia.org/wiki/OpenSSL
8. http://en.wikipedia.org/wiki/Adaptive_chosen-ciphertext_attack
9. http://en.wikipedia.org/wiki/Dual_EC_DRBG
10. Simple Backdoors for RSA Key Generation, Claude Crépeau, Alain Slakmon