Greatest Common Divisors: Attacks on RSA and Post-Quantum Security

Martin R. Albrecht **@martinralbrecht**

22/08/2016 — Big-O London

Outline

Greatest Common Divisors

Euclidean algorithm

Given two integers $a, b < N = 2^\kappa$ the Euclidean algorithm computes their greatest common divisor $\gcd(a, b)$.

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

- The Euclidean algorithm runs in time $\mathcal{O}\left(\kappa^2\right)$.
- Best known algorithm runs in time $\mathcal{O}\left(\kappa \log^2 \kappa \log\log \kappa\right)$.[1]

[1] Damien Stehlé and Paul Zimmermann. A Binary Recursive Gcd Algorithm. In: *Algorithmic Number Theory, 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 13-18, 2004, Proceedings*. Ed. by Duncan A. Buell. Vol. 3076. Lecture Notes in Computer Science. Springer, 2004, pp. 411–425. DOI: 10.1007/978-3-540-24847-7_31. URL: http://dx.doi.org/10.1007/978-3-540-24847-7_31.

RSA

Public key encryption

KeyGen Bob sends padlock pk to Alice and keeps the key sk.

Enc Alice inserts message m in a box and locks it with pk.

Dec Bob opens the box c with key sk to the padlock pk.

Public key encryption

KeyGen Bob generates a key pair (sk, pk) and publishes pk.

Enc Alice uses pk to encrypt message m for Bob as c.

Dec Bob uses sk to decrypt c to recover m.

Naive RSA

KeyGen The public key is (N, e) and the private key is d, with

- $N = p \cdot q$ where p and q prime,
- e coprime to $\varphi(N) = (p-1)(q-1)$ and
- d such that $e \cdot d = 1 \mod \varphi(N)$.

Enc $c = m^e \mod N$

Dec $m = c^d = m^{e \cdot d} = m^1 \mod N$

Naive RSA is not IND-CCA secure

- Assume we want to decrypt $c = m^e \bmod N$ with access to an oracle which will decrypt any ciphertext but c.

[2] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 1–12.

Naive RSA is not IND-CCA secure

- Assume we want to decrypt $c = m^e \bmod N$ with access to an oracle which will decrypt any ciphertext but c.
- Pick a random $s \bmod N$ and compute $c' = s^e \cdot c \bmod N$

[2] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 1–12.

Naive RSA is not IND-CCA secure

- Assume we want to decrypt $c = m^e \bmod N$ with access to an oracle which will decrypt any ciphertext but c.
- Pick a random $s \bmod N$ and compute $c' = s^e \cdot c \bmod N$
- Submit c' to the decryption oracle to recover $m' = (s^e \cdot c)^d$

[2] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 1–12.

Naive RSA is not IND-CCA secure

- Assume we want to decrypt $c = m^e \bmod N$ with access to an oracle which will decrypt any ciphertext but c.
- Pick a random $s \bmod N$ and compute $c' = s^e \cdot c \bmod N$
- Submit c' to the decryption oracle to recover $m' = (s^e \cdot c)^d$
- It holds that

$$m' = (s^e \cdot c)^d = (s^e \cdot m^e)^d = \left((s \cdot m)^e\right)^d = s \cdot m \bmod N$$

[2] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 1–12.

Naive RSA is not IND-CCA secure

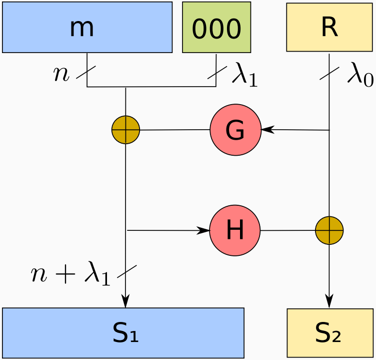
- Assume we want to decrypt $c = m^e \bmod N$ with access to an oracle which will decrypt any ciphertext but c.
- Pick a random $s \bmod N$ and compute $c' = s^e \cdot c \bmod N$
- Submit c' to the decryption oracle to recover $m' = (s^e \cdot c)^d$
- It holds that

$$m' = (s^e \cdot c)^d = (s^e \cdot m^e)^d = \left((s \cdot m)^e\right)^d = s \cdot m \bmod N$$

- Such an oracle can essentially be instantiated using error messages.[2]

[2] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 1–12.

RSA-OAEP



Use RSA-OAEP (also sometimes called ”PKCS#1 v2.1 encryption”).

“We use RSA!”

boxcryptor

Here is how it works:
Boxcryptor creates a virtual drive on your computer that allows you to encrypt your files locally before uploading them to your cloud or clouds of choice. It encrypts individual files - and does not create containers. Any file dropped into an encrypted folder within the Boxcryptor drive will get automatically encrypted before it is synced to the cloud. To protect your files, Boxcryptor uses the AES-256 and RSA encryption algorithms.

telegram

Q: So how do you encrypt data?

We support two layers of secure encryption. Server-client encryption is used in Cloud Chats (private and group chats), Secret Chats use an additional layer of client-client encryption. All data, regardless of type, is encrypted in the same way — be it text, media or files.

Our encryption is based on 256-bit symmetric AES encryption, RSA 2048 encryption, and Diffie–Hellman secure key exchange. You can find more info in the Advanced FAQ.

sicher

How can I be sure my messages are sent securely?

Sicher is using point-to-point encryption, based on asymmetric cryptography. It means that only the recipient who owns the private key can decrypt the message. RSA cryptosystem is used with 2048 bit keys. Additionally all data exchange between mobile apps and Sicher servers is protected using SSL.

back to top

Classical attacks on RSA

- An adversary who can factor large integers can break RSA.
- The best known classical algorithm for factoring is the Number Field Sieve (NFS)
- It has a super-polynomial but sub-exponential (in $\log N$) complexity of

$$\mathcal{O}\left(e^{1.9(\log^{1/3} N)(\log\log^{2/3} N)}\right)$$

operations.

Classical attacks on RSA

- An adversary who can factor large integers can break RSA.
- The best known classical algorithm for factoring is the Number Field Sieve (NFS)
- It has a super-polynomial but sub-exponential (in $\log N$) complexity of

$$\mathcal{O}\left(e^{1.9(\log^{1/3} N)(\log\log^{2/3} N)}\right)$$

operations.

Warning

This does not mean an adversary **has** to factor to solve RSA.

The GCD attack on bad random numbers

Much randomness

- When we generate RSA moduli, we need to sample two good prime numbers of bitsize $\kappa/2$
- The probability that a random number of bitsize $\kappa/2$ is prime, is about $1/\kappa$.
- To sample an RSA modulus we hence need about κ^2 random bits. For $\kappa = 1024$ this means about 10^6 random bits.
- Where do we get all these bits from?

Collecting entropy

Random bits can be gathered from the environment using various sensors, e.g.

- time,
- process IDs currently running on the machine,
- the harddisk,
- the content of uninitialised memory,
- hardware sensors (temperature etc.).

What could possibly go wrong?

Assume a router generating RSA moduli on booting for the first time.

- It might not know the time but retrieve it once booted.
- Whenever it boots the same processes are running.
- The harddisk has the same files on it for every router.
- Uninitialised memory is just full of zeros.
- There are perhaps no hardware sensors.

All routers of the same make might (in fact, some do) generate the same RSA modulus.

What could possibly go wrong?

What if two routers generate moduli $N_0 = q_0 \cdot p$ and $N_1 = q_1 \cdot p$, i.e. moduli with shared factors, due to bad randomness?

- We assume that factoring each of N_0 or N_1 is hard.
- However, computing $\gcd(N_0, N_1)$ reveals p but costs only $\mathcal{O}\left(\log^2 N\right)$ operations.

What could possibly go wrong?

What if two routers generate moduli $N_0 = q_0 \cdot p$ and $N_1 = q_1 \cdot p$, i.e. moduli with shared factors, due to bad randomness?

- We assume that factoring each of N_0 or N_1 is hard.
- However, computing $\gcd(N_0, N_1)$ reveals p but costs only $\mathcal{O}\left(\log^2 N\right)$ operations.

If only we could compute the pairwise GCD of all RSA moduli on the Internet …

[W]e are able to compute the private keys for 64,000 (0.50%) of the TLS hosts and 108,000 (1.06%) of the SSH hosts from our scan data alone by exploiting known weaknesses of RSA and DSA when used with insufficient randomness.[3]

[3] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In: *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012.* Ed. by Tadayoshi Kohno. USENIX Association, 2012, pp. 205–220.

Computing pairwise GCDs efficiently

- Naively, we'd have to compute $\mathcal{O}\left(t^2\right)$ GCDs to check all t moduli against each other.

Computing pairwise GCDs efficiently

- Naively, we'd have to compute $\mathcal{O}\left(t^2\right)$ GCDs to check all t moduli against each other.
- We can do better by performing t GCD computations

$$\gcd(N_i, \prod_{j \neq i} N_j)$$

Computing pairwise GCDs efficiently

- Naively, we'd have to compute $\mathcal{O}\left(t^2\right)$ GCDs to check all t moduli against each other.
- We can do better by performing t GCD computations

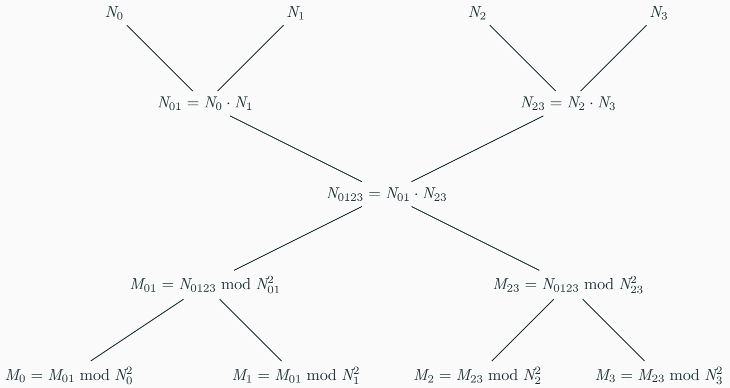
$$\gcd(N_i, \prod_{j \neq i} N_j)$$

- We will use the identity

$$x \bmod N_0 = (x \bmod N_0 \cdot N_1) \bmod N_0$$

Computing pairwise GCDs efficiently

Let, for example, $t = 4$.



- Compute $R_1 = \gcd(M_1/N_1, N_1), \ldots, R_4 = \gcd(M_4/N_4, N_4)$
- Cost: $\mathcal{O}\left(t \cdot \kappa \cdot \log^2(t \cdot \kappa) \log\log(t \cdot \kappa)\right)$

The Approximate GCD problem

Quantum attacks on RSA

An adversary with access to a quantum computer with

$$\mathcal{O}\left(\log^2(N)\log\log(N)\log\log\log(N)\right)$$

gates can factor N using Shor's algorithm.[4]

[4] http://www.scottaaronson.com/blog/?p=208

Quantum attacks on RSA

The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE NETWORKS SECURITY INFRASTRUCTURE DEVOPS BUSINESS HARDWARE SCIENCE BOOTNOTES FORUMS

Security

NIST readies 'post-quantum' crypto competition

Are you Shor you want to try this?

4 May 2016 at 05:56, Richard Chirgwin 41 37

Your mission, should you choose to accept it, is to help the National Institute of Standards and Technology (NIST) defend cryptography against the onslaught of quantum computers.

It hasn't happened yet, but it's pretty widely agreed that quantum computers pose a significant risk to cryptography. All that's needed is either a quantum computer specifically built to implement Shor's algorithm (which sets out how to factor integers using quantum computers); or a truly quantum Turing machine that can be programmed to run whatever program it's asked to run.

More like this

Cryptography Nist

Most read

Why Oracle will win its Java copyright case – and why you'll be glad when it does

Even in remotest Africa, Windows 10 nagware ruins your day: Update burns satellite link cash

UK Home Office is creating mega database by stitching together ALL its gov records

UCLA shooter: I killed my prof over code theft

BOFH: What's your point, caller?

The Approximate GCD problem is the problem of distinguishing

$$x_i = q_i \cdot p + r_i$$

from uniform $\mathbb{Z} \cap [0, X)$ with $x_i < X$.

The Approximate GCD problem

$$x_i = q_i \cdot p + r_i$$

If λ is our security parameter (think $\lambda = 128$), then

name	sizeof	DGHV10[5]	CheSte15[6]
γ	x_i	λ^5	$\lambda \log \lambda$
η	p	λ^2	$\lambda + \log \lambda$
ρ	r_i	λ	λ

[5] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43.

[6] Jung Hee Cheon and Damien Stehlé. Fully Homomophic Encryption over the Integers Revisited. In: *EUROCRYPT 2015, Part I*. ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 513–536. DOI: `10.1007/978-3-662-46800-5_20`.

KeyGen The public key is $\{x_i = q_i \cdot p + 2\, r_i\}_{0 \leq i < t}$ and the private key is p.

Enc For $m \in \{0,1\}$ output $c = m + \sum b_i \cdot x_i$ with $b_i \leftarrow_\$ \{0,1\}$.

Dec $m = (c \bmod p) \bmod 2$.

[7] In contrast to naive RSA, this scheme offers indistinguishability security under chosen plaintext attacks (IND-CPA).

Naive encryption

KeyGen The public key is $\{x_i = q_i \cdot p + 2\, r_i\}_{0 \le i < t}$ and the private key is p.

Enc For $m \in \{0, 1\}$ output $c = m + \sum b_i \cdot x_i$ with $b_i \leftarrow_{\$} \{0, 1\}$.

Dec $m = (c \bmod p) \bmod 2$.

Warning

This encryption scheme has the same malleability property as naive RSA encryption![7]

[7] In contrast to naive RSA, this scheme offers indistinguishability security under chosen plaintext attacks (IND-CPA).

Attacks on the Approximate GCD problem

Exhaustive search

Given $x_0 = q_0 \cdot p + r_0$ and $x_1 + q_1 \cdot p + r_1$ we know that

$$p = \gcd\left((x_0 - r_0), (x_1 - r_1)\right)$$

Guess r_0 and r_1!

Cost
$2^{2\rho}$ GCDs

Exhaustive search + multiplication

Compute

$$\gcd\left(x_0', \prod_{i=0}^{2^\rho-1} (x_1 - i) \bmod x_0'\right)$$

for all $x_0' = x_0 - j$ with $0 \leq j < 2^{\rho-1}$.

Cost

2^ρ GCDs, $2^{2\rho}$ multiplications

- We can reduce multiplications to $2^{\rho/2}$ per guess of x_0'.
- Define univariate polynomials mod x_0':

$$f_j(x) = \prod_{i=0}^{j-1}(x_1 - (x+i)) \in \mathbb{Z}_{x_0'}[x]$$

- Note that

$$\prod_{i=0}^{2^\rho - 1}(x_1 - i) = \prod_{k=0}^{2^{\rho/2}-1} f_{2^{\rho/2}}(2^{\rho/2}k)$$

Example

- $\rho = 2$, $f_2 = (x_1 - (x+0)) \cdot (x_1 - (x+1))$
- $f_2(0) \cdot f_2(2) = (x_1 - 0) \cdot (x_1 - 1) \cdot (x_1 - 2) \cdot (x_1 - 3)$

Time-memory trade-off

Compute

$$\gcd\left(x_0', \prod_{k=0}^{2^{\rho/2}-1} f_{2^{\rho/2}}(2^{\rho/2}k) \bmod x_0'\right)$$

for all $x_0' = x_0 - j$ with $0 \leq j < 2^{\rho-1}$.

Cost

- 2^ρ GCDs and computation of $f_{2^{\rho/2}}(x) \bmod x_0'$,
- per guess for x_0': $2^{\rho/2}$ multiplications and evaluations of $f_{2^{\rho/2}}(x)$.

- Computing $f_{2^{\rho/2}}(x) \bmod x_0'$ can be accomplished in time $\mathcal{O}\left(2^{\rho/2} \cdot \rho\right)$ using the Fast Fourier Transform.
- Evaluating $f_{2^{\rho/2}}(x) \bmod x_0'$ at our $2^{\rho/2}$ points can be accomplished in time $\mathcal{O}\left(2^{\rho/2} \cdot \rho\right)$ using the Fast Fourier Transform.
- The strategy is similar to the pairwise GCD case earlier

Cost

$2^{\mathcal{O}\left(3/2\rho \log^2 \rho\right)}$ operations.[8]

[8] Yuanmi Chen and Phong Q. Nguyen. Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 502–519.

Lattice attacks

Given $x_0 = q_0 p + r_0$ and $x_1 = q_1 p + r_1$, consider

$$\begin{aligned} q_0 x_1 - q_1 x_0 &= q_0(q_1 p + r_1) - q_1(q_0 p - r_0) \\ &= q_0 q_1 p + q_0 r_1 - q_1 q_0 p - q_1 r_0 \\ &= q_0 r_1 - q_1 r_0 \end{aligned}$$

and note that

$$q_0 x_1 - q_1 x_0 \ll x_i$$

Given $x_0 = q_0 p + r_0$ and $x_1 = q_1 p + r_1$, consider

$$
\begin{aligned}
q_0 x_1 - q_1 x_0 &= q_0(q_1 p + r_1) - q_1(q_0 p - r_0) \\
&= q_0 q_1 p + q_0 r_1 - q_1 q_0 p - q_1 r_0 \\
&= q_0 r_1 - q_1 r_0
\end{aligned}
$$

and note that

$$q_0 x_1 - q_1 x_0 \ll x_i$$

Non-starter?

We don't know q_i!

Consider the matrix

$$\mathbf{B} = \begin{pmatrix} 2^{\rho+1} & x_1 & x_2 & \cdots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \ddots & \\ & & & & -x_0 \end{pmatrix}$$

multiplying on the left by the vector $\mathbf{q} = (q_0, q_1, q_2, \cdots, q_t)$ gives

$$\begin{aligned} \mathbf{v} &= (q_0, q_1, \cdots, q_t) \cdot \mathbf{B} \\ &= (q_0\, 2^{\rho+1}, q_0 x_1 - q_1 x_0, \cdots, q_0 x_t - q_t x_0) \\ &= (q_0\, 2^{\rho+1}, q_0 r_1 - q_1 r_0, \cdots, q_0 r_t - q_t r_0) \end{aligned}$$

which is a vector with small coefficients compared to x_i.

We call the set of all integer-linear combinations of the rows of $\mathbf{B}$ the lattice spanned by (the rows of) $\mathbf{B}$.

SVP finding a shortest non-zero vector on general lattices is NP-hard.

Gap-SVP finding short non-zero vectors on general lattices is a well-known and presumed quantum-hard problem.

Easy SVP

GCD is SVP on the integer lattice $\mathbb{Z}$. For example, $\mathbf{B} = [21, 14]^T$, $\mathbf{v} = (-1, 1)$, $\mathbf{v} \cdot \mathbf{B} = 7$.

Reduction to lattice problem

We can show that an adversary has to solve Gap-SVP.

AGCD → LWE

If there is an algorithm efficiently solving the AGCD problem then there exists an algorithm which solves the **Learning with Errors** (LWE) problem with essentially the same performance.[9]

LWE → Gap-SVP

If there is an algorithm efficiently solving the LWE problem then there exists a quantum algorithm which solves worst-case Gap-SVP instances.[10]

[9] Jung Hee Cheon and Damien Stehlé. Fully Homomophic Encryption over the Integers Revisited. In: *EUROCRYPT 2015, Part I*. ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 513–536. DOI: `10.1007/978-3-662-46800-5_20`.

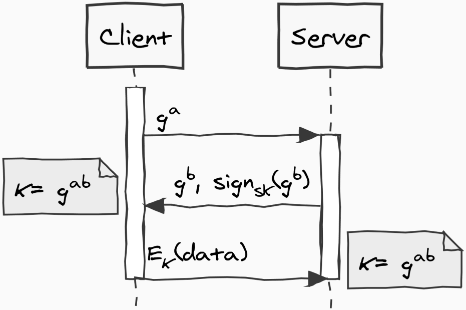
[10] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In: *37th ACM STOC*. ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93.

Google's post-quantum experiment: "A New Hope"

Ring-LWE

- The Learning with Errors problem is essentially the problem of solving a linear system of equations in the presence of noise.
- Given $\mathbf{A}, \mathbf{c}$ solve
$$\mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q$$
for $\mathbf{s}$ when $\mathbf{e}$ is “small”.
- The matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is kinda big.
- To make it smaller, use structured matrices, e.g. negacyclic matrices $\Rightarrow$ Ring-LWE.

A New Hope[11] : Ring-LWE based key exchange



[11] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092. `http://eprint.iacr.org/2015/1092`. 2015.

Thank you

Questions?

Bonus

Given $c_i = q_i \cdot p + m'_i$ with $m'_i = 2\,r_i + m_i$.

- We can compute

$$c' = c_0 \cdot c_1 = q_0\,q_1\,p^2 + q_0\,m'_1\,p + q_1\,m'_0\,p + m'_0 \cdot m'_1$$

  to get $c' \bmod p = m'_0 \cdot m'_1$ and $m'_0 \cdot m'_1 \bmod 2 = m_0 \cdot m_1$.

- We can also compute

$$c' = c_0 + c_1 = (q_0 + q_1)p + (m'_0 + m'_1)$$

  to get $c' \bmod p \bmod 2 = m_0 \oplus m_1$.

We can compute with encrypted data.[12]

[12] https://crypto.stanford.edu/craig/easy-fhe.pdf