

GREATEST COMMON DIVISORS: ATTACKS ON RSA AND POST-QUANTUM SECURITY

Martin R. Albrecht @martinralbrecht

13/11/2017 — TU Graz

OUTLINE

Greatest Common Divisors

RSA

The GCD attack on bad random numbers

The Approximate GCD problem

Attacks on the Approximate GCD problem

Bonus

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 $\mathcal{O}(\kappa^2)$.

Best known algorithm runs in time $\mathcal{O}(\kappa \log^2 \kappa \log \log \kappa)$.¹

For comparison, integer multiplication costs $\mathcal{O}(\kappa \log \kappa \log \log \kappa)$ using the Schönhage–Strassen algorithm.

¹Damien 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 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 $\phi(N) = (p - 1)(q - 1)$ and
- d such that $e \cdot d \equiv 1 \pmod{\phi(N)}$.

Enc $c \equiv m^e \pmod{N}$

Dec $m \equiv c^d \equiv m^{e \cdot d} \equiv m^1 \pmod{N}$

NAIVE RSA IS NOT IND-CCA SECURE

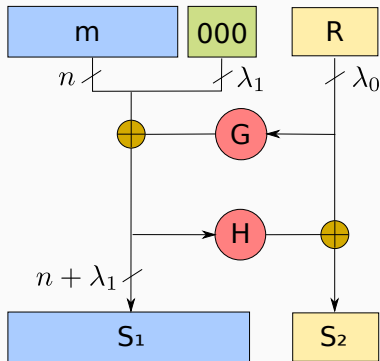
- Assume we want to decrypt $c \equiv m^e \pmod N$ with access to an oracle which will decrypt any ciphertext but c .
- Pick a random $s \pmod N$ and compute $c' \equiv s^e \cdot c \pmod N$
- Submit c' to the decryption oracle to recover $m' \equiv (s^e \cdot c)^d$
- It holds that

$$m' \equiv (s^e \cdot c)^d \equiv (s^e \cdot m^e)^d \equiv ((s \cdot m)^e)^d \equiv s \cdot m \pmod N$$

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

²Daniel 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").

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.

Caution

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?

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.).
- your phone's camera is an excellent source of randomness.³

³<https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/>

WHAT COULD POSSIBLY GO WRONG?

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

- It might not know the time but retrieves after boot.
- 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?

If two routers compute the same $N = p \cdot q$ then their respective users can read each other's traffic.

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.
- On the other hand, computing $\gcd(N_0, N_1)$ reveals p but costs only $\mathcal{O}(\kappa \log^2 \kappa \log \log \kappa)$ operations when $N_i \approx 2^\kappa$.

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.
- On the other hand, computing $\gcd(N_0, N_1)$ reveals p but costs only $\mathcal{O}(\kappa \log^2 \kappa \log \log \kappa)$ operations when $N_i \approx 2^\kappa$.

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

THE GCD ATTACK ON POOR RANDOM NUMBERS

[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.⁴

⁴Nadia 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.

GRABBING ALL RSA MODULI

```
malb@computer:: ~ > sudo zmap -p 22 -r 10000 -i eth0 -o ssh.csv
malb@computer:: ~ > tail -n +2 ssh.csv | zgrab --port=22 --ssh=true --output-file='ssh.json'
...
{"ip": "REDACTED",
 "data": {"ssh": {
   "server_protocol": {
     "raw_banner": "SSH-2.0-OpenSSH_6.7p1 Debian-5\r\n",
     "protocol_version": "2.0",
     "software_version": "OpenSSH_6.7p1",
     "comments": "Debian-5"}, ...}}}
```

<https://zmap.io>

COMPUTING PAIRWISE GCDs EFFICIENTLY

- Naively, we'd have to compute $\mathcal{O}(\tau^2)$ GCDs to check all τ moduli against each other.

⁵<https://factorable.net/fastgcd-1.0.tar.gz>

COMPUTING PAIRWISE GCDs EFFICIENTLY

- Naively, we'd have to compute $\mathcal{O}(\tau^2)$ GCDs to check all τ moduli against each other.
- We can do better by performing τ GCD computations

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

⁵<https://factorable.net/fastgcd-1.0.tar.gz>

COMPUTING PAIRWISE GCDs EFFICIENTLY

- Naively, we'd have to compute $\mathcal{O}(\tau^2)$ GCDs to check all τ moduli against each other.
- We can do better by performing τ GCD computations

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

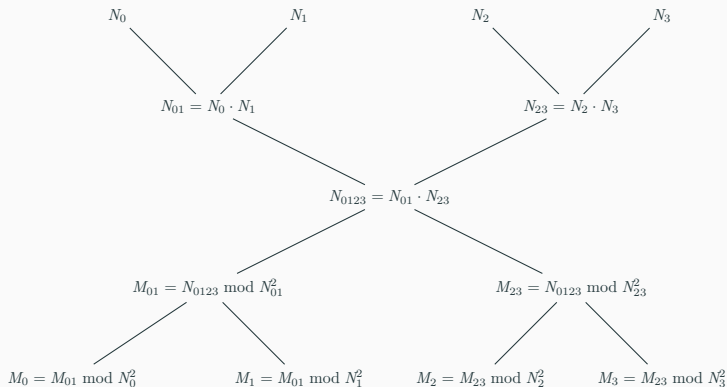
- We will use the identity⁵

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

⁵<https://factorable.net/fastgcd-1.0.tar.gz>

COMPUTING PAIRWISE GCDs EFFICIENTLY

Let, for example, $\tau = 4$.



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

THE APPROXIMATE GCD PROBLEM

An adversary with access to a quantum computer with

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

gates can factor N using Shor's algorithm.⁶

⁶Peter W. Shor. [Algorithms for Quantum Computation: Discrete Logarithms and Factoring](#). In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134.

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](#)


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

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 ⁷	CheSte15 ⁸
γ	x_i	λ^5	$\lambda \log \lambda$
η	p	λ^2	$\lambda + \log \lambda$
ρ	r_i	λ	λ

⁷Marten 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.

⁸Jung Hee Cheon and Damien Stehlé. [Fully Homomorphic 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.

NAIVE ENCRYPTION

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

⁹In contrast to naive RSA, this scheme offers indistinguishability security under chosen plaintext attacks (IND-CPA).

NAIVE ENCRYPTION

KeyGen The public key is $\{x_i = q_i \cdot p + 2 r_i\}_{0 \leq i < t}$ 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 $m = (c \bmod p) \bmod 2$.

Caution

This encryption scheme has the same malleability property as naive RSA encryption!⁹

⁹In 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

$$p \mid \gcd((x_0 - r_0), (x_1 - r_1))$$

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

TIME-MEMORY TRADE OFF

Lemma

Assume that we have τ samples $x_0, \dots, x_{\tau-1}$ of a given prime p , of the hidden form $x_i = q_i \cdot p + r_i$, then p can then be recovered with overwhelming probability in time $\tilde{O}(2^{\frac{\tau+1}{\tau-1}\rho})$.¹⁰

¹⁰Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. [Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers](#). In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 446–464.

TIME-MEMORY TRADE OFF: MULTIPLICATION

The algorithm proceeds by computing

$$y_i = \prod_{j=0}^{2^\rho - 1} (x_i - j)$$

for $0 \leq i < \tau$. Each y_i can be computed in quasilinear time in 2^ρ , i.e.

$$\mathcal{O}(2^\rho \cdot \gamma \cdot \log^2(2^\rho \cdot \gamma) \cdot \log \log(2^\rho \cdot \gamma))$$

using a product tree.

TIME-MEMORY TRADE OFF: GCD

We have

$$p \mid \gcd(y_0, \dots, y_{\tau-1}).$$

The GCD can be evaluated as

$$\gcd(\dots \gcd(\gcd(y_0, y_1), y_2), \dots, y_{\tau-1})$$

using $\tau - 1$ quasilinear GCD computations on numbers of size $\mathcal{O}(2^\rho \gamma) = \tilde{\mathcal{O}}(2^\rho)$, i.e. the total cost of this step is

$$\mathcal{O}(\tau \cdot 2^\rho \cdot \gamma \cdot \log^2(2^\rho \cdot \gamma) \cdot \log \log(2^\rho \cdot \gamma)).$$

Hence, the whole computation of the GCD takes time

$$\mathcal{O}(\tau \cdot 2^\rho \cdot \gamma \cdot \log^2(2^\rho \cdot \gamma) \cdot \log \log(2^\rho \cdot \gamma)).$$

TIME-MEMORY TRADE OFF: FILTERING

- The GCD g will be much bigger than p .
- But with high probability over the choice of the (q_i, r_i) , all the prime factors of g except p are smaller than some bound B that is not much larger than 2^ρ .
- Thus, p can be recovered as g/g' , where g' is the B -smooth part of g , which can in turn be computed in quasilinear time in $\max(B, \log g)$, e.g. using Bernstein's algorithm¹¹.

Overall, the full time complexity of the attack is thus

$$\mathcal{O}(\tau \cdot 2^\rho \cdot \gamma \cdot \log^2(2^\rho \cdot \gamma) \cdot \log \log(2^\rho \cdot \gamma))$$

assuming B has size about 2^ρ .

¹¹Dan Bernstein. [How to Find Smooth Parts of Integers](http://cr.yp.to/papers.html#smoothparts). Available at <http://cr.yp.to/papers.html#smoothparts>. 2004.

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

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

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

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

Non-starter?

We don't know q_i !

LATTICE ATTACKS

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 $\mathbf{q} = (q_0, q_1, q_2, \dots, q_t)$ gives

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

which is a vector with small coefficients compared to x_i .

FINDING SHORT VECTORS

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 $_{\gamma}$ Differentiating between instances of SVP in which the answer is at most 1 or larger than γ on **general** lattices is a well-known and presumed quantum-hard problem for γ polynomial in lattice dimension.

Easy SVP

GCD is SVP on \mathbb{Z}^2 . For example, $\mathbf{B} = [21, 14]^T$, $\mathbf{v} = (-1, 1)$, $\mathbf{v} \cdot \mathbf{B} = 7$.

REDUCTION TO PRESUMED HARD LATTICE PROBLEM

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

AGCD \rightarrow 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.¹²

LWE \rightarrow 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.¹³

¹²Jung Hee Cheon and Damien Stehlé. **Fully Homomorphic 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.

¹³Oded 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.

LEARNING WITH ERRORS (IN NORMAL FORM)

Given (\mathbf{A}, \mathbf{c}) with $\mathbf{c} \in \mathbb{Z}_q^m$, $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, small $\mathbf{s} \in \mathbb{Z}^n$ and small $\mathbf{e} \in \mathbb{Z}^m$ is

$$\begin{pmatrix} \mathbf{c} \end{pmatrix} = \begin{pmatrix} \leftarrow & n & \rightarrow \\ & \mathbf{A} & \end{pmatrix} \times \begin{pmatrix} \mathbf{s} \end{pmatrix} + \begin{pmatrix} \mathbf{e} \end{pmatrix}$$

or $\mathbf{c} \leftarrow_{\mathbf{s}} \mathcal{U}(\mathbb{Z}_q^m)$.

FROM VECTORS TO SCALARS

LWE with modulus q^n and dimension 1 is as hard as LWE with modulus q and dimension 1.

$$q^{d-1} \cdot \langle \mathbf{a}, \mathbf{s} \rangle \approx \left(\sum_{i=0}^{n-1} q^i \cdot a_i \right) \cdot \left(\sum_{i=0}^{d-1} q^{d-i-1} \cdot s_i \right) \bmod q^d = \tilde{a} \cdot \tilde{s} \bmod q^d.$$

Example

$$\begin{aligned} (a_0 + q \cdot a_1) \cdot (q \cdot s_0 + s_1) &= q(a_0 \cdot s_0 + a_1 \cdot s_1) + (a_1 \cdot s_1) + q^2(a_1 \cdot s_0) \\ &\equiv q(a_0 \cdot s_0 + a_1 \cdot s_1) + (a_1 \cdot s_1) \bmod q^2 \\ &\approx q(a_0 \cdot s_0 + a_1 \cdot s_1) \bmod q^2 \end{aligned}$$

FIN



QUESTIONS?

BONUS

HOMOMORPHIC ENCRYPTION

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.¹⁴

¹⁴<https://crypto.stanford.edu/craig/easy-fhe.pdf>