



Norwegian University of
Science and Technology

RANDOMNESS 3: RANDOMIZATION

TTM4205 – Lecture 5

Tjerand Silde

28.08.2026

Contents

Primality Testing

Factorization

De-Randomization

Takeaways

Contents

Primality Testing

Factorization

De-Randomization

Takeaways

Reference Material

These slides are based on:

- ▶ The referred papers in the slides
- ▶ JPA: parts of chapter 9

How do we check if
a number is prime?

Deterministic Methods

- ▶ Brute force
- ▶ Sieving methods
- ▶ Wilson's theorem?
- ▶ Fermat's theorem?

Brute Force Testing

It is always possible to test every candidate divisor, but how long does it take?
Assume that p is of 2048 bits.

- ▶ Is p divisible by any number between 1 and p ? $\sim 2^{2048}$ trials
- ▶ By 2 or any odd number between 1 and p ? $\sim 2^{2047}$ trials
- ▶ By 2 or any odd number between 1 and $\sqrt{p}$? $\sim 2^{1023}$ trials

This is infeasible: even 2^{128} operations are considered out of reach.

Sieving Methods

It is possible to pre-compute many small prime numbers to speed up the process, e.g., the sieve of Eratosthenes:

First, keep 2 and remove all even numbers. Then, keep 3 and remove all multiples of 3. Now 4 is already removed. Keep 5 and remove all multiples of 5, and 6 has already been removed, and so on.

This still requires exponential work to rule out every divisor.

Wilson's Theorem

Wilson's Theorem: *A natural number $p > 1$ is a prime number if and only if the product of all the positive integers less than p is one less than a multiple of p .*

This means: $(p - 1)! \equiv -1 \pmod{p} \iff p \text{ is a prime number.}$

It still requires exponential work to compute $(p - 1)!$

Similar techniques can speed this up, but it remains impractical.

Randomized Methods

- ▶ Monte Carlo algorithms
- ▶ The Miller-Rabin method

Monte Carlo Algorithms

A Monte Carlo algorithm is a randomized algorithm whose output may be incorrect with a given probability.

A **false**-biased Monte Carlo algorithm is always correct when it returns **false**, but it might be wrong if it outputs **true**. Opposite for a **true**-biased algorithm.

If the probability of being wrong is $1/2$, then repeating the test with fresh randomness drives it down: λ independent rounds give error $(\frac{1}{2})^\lambda \rightarrow 0$.

Some commonly used algorithms: Solovay-Strassen and Miller-Rabin.

The Fermat Test and Carmichael Numbers

Fermat's little theorem: if p is prime, then $a^{p-1} \equiv 1 \pmod{p}$ for every a not divisible by p .

The Fermat test turns this around: sample a , and if $a^{p-1} \not\equiv 1 \pmod{p}$, then p is certainly composite. If the congruence holds, we call p a probable prime.

This is sound in one direction only, and there are composites that satisfy the congruence for *every* base a coprime to them. These are the *Carmichael numbers*, the smallest being $561 = 3 \cdot 11 \cdot 17$, followed by 1105, 1729, and 2465.

The Fermat Test and Carmichael Numbers

Korselt's criterion: a composite p is a Carmichael number if and only if it is square-free and $(q - 1) \mid (p - 1)$ for every prime $q \mid p$. For 561 the values 2, 10 and 16 all divide 560.

They are rare: there are 7 below 10^4 and 16 below 10^5 .

They are not rare enough to ignore, however. Alford, Granville, and Pomerance proved in 1994 that there are infinitely many of them, and an adversary who gets to choose the modulus can simply construct one.

The Fermat Test and Carmichael Numbers

Repetition does not help against a Carmichael number, since every coprime base passes. The test only fails when we happen to pick an a sharing a factor with p , and then $\gcd(a, p)$ hands us the factorization directly.

This is why the Fermat test is never used on its own for cryptographic key generation, and why we want a test with no analogue of the Carmichael numbers: every composite must have many witnesses, whichever base the adversary hopes we will pick.



Miller-Rabin Primality Testing

Miller-Rabin strengthens the Fermat test by also using the square roots of 1: modulo a prime, the only ones are ± 1 .

Let p be an odd integer and write $p - 1$ as $2^s \cdot d$ where d is an odd number.

Let $1 < a < p$ be randomly sampled. If p is prime, then $a^d \equiv 1 \pmod p$ or $a^{2^r d} \equiv -1 \pmod p$ for some $0 \leq r < s$, at most $\log_2 p$ options.

Miller-Rabin Primality Testing

In practice: compute $a^d \bmod p$ and square it up to $s - 1$ times. If we never see 1 or -1 , then p is composite and a is a *witness* to that.

If p is composite, at most $\frac{1}{4}$ of the bases fail to be witnesses, so λ fresh bases leave at most a $(\frac{1}{4})^\lambda$ chance of accepting it. This holds for *every* composite: there is no analogue of the Carmichael numbers here.

For a randomly chosen candidate the error is far smaller, which is why 40 rounds are more than enough in practice.

Primality Testing APIs

In practice, the primality of a candidate p is checked by combining the methods above:

1. Pre-compute a list of the first thousand prime numbers.
2. Reject p if it is divisible by any prime number in the list.
3. Run the Miller-Rabin algorithm, say, ~ 40 times.
4. If all checks succeed, then output: *probably prime*.

Primality Testing Failures

It might be highly rewarding for an attacker to convince you that a composite number is prime. Why is that?

If a factor of an RSA modulus is not actually prime, then n has smaller factors than intended and is far easier to factor.

Pohlig-Hellman likewise computes discrete logarithms easily in a group of composite order, unless the order has a large prime factor.

A classic mistake in Miller-Rabin: the bases a are not sampled per call, but hard-coded, or derived deterministically from the candidate. An attacker who knows which bases will be used can *construct* composites that pass all of them.

Primality Testing in OpenSSL

Prime and Prejudice: Primality Testing Under Adversarial Conditions

Martin R. Albrecht¹, Jake Massimo¹, Kenneth G. Paterson¹, and Juraj Somorovsky²

¹ Royal Holloway, University of London

² Ruhr University Bochum, Germany

`martin.albrecht@rhul.ac.uk`, `jake.massimo.2015@rhul.ac.uk`, `kenny.paterson@rhul.ac.uk`,
`juraj.somorovsky@rub.de`

Figure: <https://eprint.iacr.org/2018/749.pdf>



The Need for Secure Primality Testing

Safety in Numbers: On the Need for Robust Diffie-Hellman Parameter Validation

Steven Galbraith¹, Jake Massimo², and Kenneth G. Paterson²

¹ University of Auckland

² Royal Holloway, University of London

s.galbraith@auckland.ac.nz, jake.massimo.2015@rhul.ac.uk,

kenny.paterson@rhul.ac.uk

Figure: <https://eprint.iacr.org/2019/032.pdf>



Secure Primality Testing API

A Performant, Misuse-Resistant API for Primality Testing

Jake Massimo¹ and Kenneth G. Paterson²

¹ Information Security Group,
Royal Holloway, University of London
`jake.massimo.2015@rhul.ac.uk`

² Department of Computer Science,
ETH Zurich
`kenny.paterson@inf.ethz.ch`

Figure: <https://eprint.iacr.org/2020/065.pdf>



Contents

Primality Testing

Factorization

De-Randomization

Takeaways

How do we factor
large bi-primes?

Deterministic Methods

Some trivial ways to attack an RSA modulus n :

- ▶ Brute force by checking whether n is divisible by 2 or by any odd number less than $\sqrt{n}$. This requires exponential work.
- ▶ Even checking only the primes between 2^{1023} and 2^{1024} , for a 2048-bit n , requires exponential work.
- ▶ Fermat factorization finds the factors quickly when p and q are close to each other, and hence both close to $\sqrt{n}$.
- ▶ Pollard's rho finds a factor p in time $\approx \sqrt{p}$, which is $\sqrt[4]{n}$ for a balanced bi-prime: it is fast only when one factor is small.

Randomness comes to our rescue in this situation as well.

Randomized Methods

The Number Field Sieve is the most efficient known algorithm for factorization, and like the sieve algorithms before it, it builds a congruence of squares.

Collect many relations $x_i^2 \equiv y_i \pmod n$ where the y_i are *smooth*, that is, they factor completely over a fixed set of small primes.

Linear algebra over $\mathbb{F}_2$ on the exponent vectors then combines relations into $a^2 \equiv b^2 \pmod n$, which means that $a^2 - b^2 \equiv (a - b)(a + b) \equiv 0 \pmod n$.

We *might* then find a factor of n by computing the greatest common divisor of n with $a - b$ and with $a + b$. This fails when $a \equiv \pm b$, which happens about half of the time, in which case we combine another set of relations and try again.

Number Field Sieve

The running time of the Number Field Sieve is

$$\exp\left((64/9)^{1/3}(\log n)^{1/3}(\log \log n)^{2/3}(1 + o(1))\right)$$

This algorithm is *sub-exponential*. The largest number ever factored in public is RSA-250, of 829 bits, in 2020, and it is still the record.

Factoring as a service: in 2015, a 512-bit RSA key could be factored in under four hours for roughly \$75 of rented computing power.

Note the gap: 829 bits took thousands of core-years, while 512 bits takes a few hours and a credit card.



Factoring as a Service

Luke Valenta, Shaanan Cohney, Alex Liao,
Joshua Fried, Satya Bodduluri, Nadia Heninger

University of Pennsylvania

Figure: <https://eprint.iacr.org/2015/1000.pdf>

Breaking into Parcel Boxes Today

Knakk svak nøkkel

– Vi fant mest mulig info, og samla det sammen for å lage en antakelse om hvordan dette fungerte, forklarer UnblvR.

Ett av de viktigste funnene i prosessen var at sikkerheten stod og falt på en offentlig 512-bits RSA-nøkkel – én for PostNord-boksene, én for Posten-boksene.

RSA-nøkler på bare 512 bits ble knekt allerede i 1999, og blir ikke ansett å være sikre å bruke i 2025. Det fikk utviklerne raskt se at de heller ikke er:

De leide seg en server på nett, brukte offentlig tilgjengelig programvare – og på under ett døgn, for under femtilappen, var pakkeboks-nøklerne knekt.

– Da vi trodde vi hadde skjønt alt, betalte vi Posten for å sende en pakke, og gikk til pakkeboksen for å prøve å åpne luka vi hadde fått tildelt, fra vår egen løsning, forteller UnblvR.

– Og det fikk vi til.

Figure: <https://www.kode24.no/artikkel/her-hacker-han-opp-pakkeboksen/243107>



State of the Art

The state of the art in integer factoring and breaking public key cryptography

Fabrice Boudot¹, Pierrick Gaudry², Aurore Guillevic², Nadia Heninger³, Emmanuel Thomé²,
and Paul Zimmermann²

¹Université de Limoges, XLIM, UMR 7252, F-87000 Limoges, France

²Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

³University of California, San Diego, USA

Figure: <https://hal.science/hal-03691141/document>



RSA, DH and DSA in the Wild*

Nadia Heninger

University of California, San Diego, USA

Figure: <https://eprint.iacr.org/2022/048.pdf>

Fermat Factorization in the Wild

Hanno Böck

January 8, 2023

Figure: <https://eprint.iacr.org/2023/026.pdf>

Ron was wrong, Whit is right

Arjen K. Lenstra¹, James P. Hughes²,
Maxime Augier¹, Joppe W. Bos¹, Thorsten Kleinjung¹, and Christophe Wachter¹

¹ EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland

² Self, Palo Alto, CA, USA

Figure: <https://eprint.iacr.org/2012/064.pdf>

Shared Prime Factors

Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices

Nadia Heninger^{†*}

Zakir Durumeric^{‡*}

Eric Wustrow[‡]

J. Alex Halderman[‡]

[†] *University of California, San Diego*
nadiah@cs.ucsd.edu

[‡] *The University of Michigan*
{zakir, ewust, jhalderm}@umich.edu

Figure: Check out the blog post, paper and slides: 1)

<https://freedom-to-tinker.com/2012/02/15/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs>, 2)

<https://factorable.net/weakkeys12.extended.pdf>, 3)

<https://crypto.stanford.edu/RealWorldCrypto/slides/nadia.pdf>



Contents

Primality Testing

Factorization

De-Randomization

Takeaways

De-Randomized Crypto

Key generation always needs randomness: there is nothing to derive it from.

Encryption does too. A deterministic public-key encryption scheme can never be IND-CPA secure, since equal plaintexts give equal ciphertexts, and the adversary sees that.

But everything that is keyed *after* key generation can derive its randomness from the secret key instead of sampling it.

De-Randomized Crypto

- ▶ Derive the per-message randomness from the key: $r = H(\text{sk}, m)$ for Schnorr, which is RFC 6979 for (EC)DSA and built into Ed25519.
- ▶ A counter, a master seed, and a hash function, so that each call obtains a fresh value without asking the operating system.
- ▶ HMAC is a deterministic MAC, so no randomness is needed at all.
- ▶ Nonce-based encryption: the nonce must be unique, not unpredictable.
- ▶ Hedging: mix the derived value with whatever fresh randomness is available, so the result is no worse than either source alone.

Hedged Public-Key Encryption: How to Protect Against Bad Randomness

Mihir Bellare* Zvika Brakerski[†] Moni Naor[‡] Thomas Ristenpart[§]
Gil Segev[¶] Hovav Shacham^{||} Scott Yilek^{**}

April 21, 2012

Figure: <https://www.cs.utexas.edu/~hovav/dist/hedge.pdf>

Contents

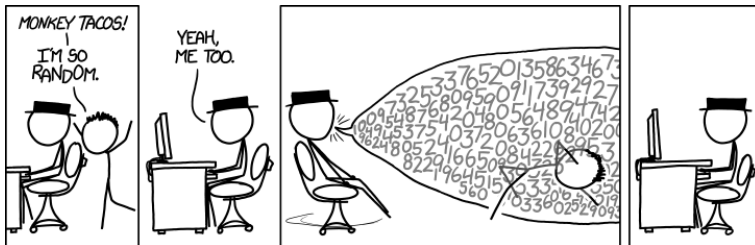
Primality Testing

Factorization

De-Randomization

Takeaways

I Am So Random



Random Number Generation

Check the quality of the built-in RNG that you rely on:

- ▶ How does it collect randomness?
- ▶ Is the RNG seeded, or pre-seeded?
- ▶ How much entropy does it provide?
- ▶ Does it warn you about issues?
- ▶ Is it cryptographically secure?



Pseudo-Random Number Generation

Check the quality of the built-in PRNG that you rely on:

- ▶ Does it rely on a proper RNG as seed? Is it pre-seeded?
- ▶ Is the PRNG cryptographically secure? NIST-approved?
- ▶ Verify the output: Do values repeat? Correct bit-size?
- ▶ Which library/version is used? Known vulnerabilities?

Some good resources are available at

<https://github.com/veorq/cryptocoding#use-strong-randomness>.

A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

Figure: <https://csrc.nist.gov/pubs/sp/800/22/r1/upd1/final>

Statistical tests catch broken sources, but they never certify a cryptographic one: a counter encrypted under AES passes all of them.

Choice of Primitives

Check the cryptographic primitive that you rely on:

- ▶ Does it rely on a proper PRNG? Is it pre-seeded?
- ▶ Is it the newest and most secure primitive? NIST-approved?
- ▶ Verify the output: Do values repeat? Correct bit-size?
- ▶ Which library/version is used? Known vulnerabilities?
- ▶ Are there de-randomized algorithms available instead?

Rolling Your Own Crypto

Security Cryptography Whatever

The Great "Roll Your Own Crypto" Debate with Filippo Valsorda

JULY 31, 2021 SECURITY, CRYPTOGRAPHY, WHATEVER



Figure: <https://securitycryptographywhatever.buzzsprout.com/1822302/8953842-the-great-roll-your-own-crypto-debate-with-filippo-valsorda>

What to Remember

- ▶ Randomized algorithms make primality testing practical at all
- ▶ A test is only as good as its randomness: fixed bases invite crafted pseudo-primes
- ▶ Key generation is where randomness cannot be avoided, and where it keeps failing
- ▶ Everything after key generation can be de-randomized, away from a fragile PRNG
- ▶ A derived nonce repeats, so hedge it with fresh randomness

Questions?