NTNU | Norwegian University of Science and Technology

COURSE SUMMARY

TTM4205 – Lecture 19

Tjerand Silde

06.11.2026

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

The Rest of the Semester

Date	What
Tue 10 November	Essay presentations, then exercise session (3 hours)
Fri 13 November	Essay presentations
Tue 17 November	Exercise session (4 hours)
Fri 20 November	Essay presentations
Fri 4 December	Weekly problems and lab report: December 4th at 23:59
Fri 18 December	Final essay: December 18th at 23:59

No more lectures after today. We may add extra exercise and lab sessions before 4 December; it will be announced on Canvas and the course website.

Essay Presentations

- Each presentation is about 10 minutes, followed by questions
- We do not expect a finished essay: introduce your topic, say what you will work on, and explain why it is interesting and useful
- Send your slides, made with the mandatory LaTeX template, to `tjerand.silde@ntnu.no` at least **one hour** before the session starts
- The presentation is mandatory but carries no points; moving to 10 November is welcome, since the other two days are full

Presentation Schedule

Tuesday 10 November

2: Nonce-Misuse-Resistant AE
23: Backdoors and Exceptional Access
42: Cryptography in Go

Friday 20 November

30: Static Analysis of Crypto Code
9: Side-Channel Attacks Against SDitH
40: Implementing the NTT Securely
22: Migrating TLS to PQC
7: Attacks on Disk Encryption
1: Nonce-Misuse-Resistant AE
6: Choosing Cryptographic Parameters

Friday 13 November

37: Microarchitectural Side Channels
8: Side-Channel Attacks Against MAYO
15: Padding-Oracle Attacks in Practice
26: Randomness Failures in the Wild
20: Backdoors and Kleptography
5: Cryptographic Safety in Rust
21: Cryptographic Fuzzing
13: ZK Proof Failures in Threshold ECDSA
4: Cryptographic Agility

The ChipWhisperer Lab

- The lab notebooks were updated on 3 November, and the lab description now has hints for the time-consuming tasks
- If you have not started a lab yet, pull the repository; if you have already filled in cells, keep your copy and apply the fixes in Appendix B.3 of the lab description by hand, to avoid merge conflicts in the notebooks
- Write the report in the LaTeX template, cite every resource you rely on, and remember the rules on AI in the assignment text
- The deadline is December 4th at 23:59, together with the weekly problems; use the exercise classes on 10 and 17 November, and any extra sessions we announce, to get help from the teaching assistants

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

The Aim of the Course

The goal was to show you a variety of attacks on, and mitigations for, the cryptography we use today, and to teach you to think as an attacker, so that you can protect your own systems.

We went through a lot of material. You are not supposed to remember all of it. You are expected to know what to look for, where to find out more, to have an understanding you can apply to similar problems, and to have ideas for how to protect against the attacks.

Think Like an Attacker, Build Like a Defender

- **Understand what can go wrong:** find the gap between the model a scheme is proven secure in and the system it runs in: randomness, error messages, timing and power, inputs, and keys shared across protocols
- **Exploit it:** turn the gap into an attack: a reused nonce into a key, an error message into an oracle, a power trace into a key byte, a malicious input into a forgery
- **Fix it:** close the gap: good randomness, AEAD and identical failures, constant time and masking, input validation, key separation
- **Avoid it:** use standard, well-analysed constructions and libraries, validate everything, and prove the whole system, not just its pieces

Security proofs are not (necessarily) wrong, but the model and attacker might not match, and implementations might not satisfy all assumptions.

History, Politics and Ethics

- Cryptography was long a military matter; Enigma and Crypto AG show what a broken cipher and a backdoored supplier cost
- The crypto wars never ended: export controls, key escrow, Dual EC, and now client-side scanning under Chat Control
- Analysing cryptography in the wild: pick a target, state the adversary, model or attack, then disclose responsibly with the users in mind
- Encrochat: law enforcement does not break the cryptography, it compromises the endpoint, and the courts then have to decide what that evidence is worth

Analysing Cryptography in the Wild

- **Choose the target** by its users: who depends on it, and in which situation, decides what is worth analysing
- **Set the adversary** from how the product is used, not from the vendor's marketing: if it promises end-to-end encryption, the provider is the adversary
- **Model, then attack or prove**: reduce the system to its cryptographic core, try the usual suspects first, and remember that a proof only covers its model
- **Disclose responsibly**: privately, with a deadline of about 90 days
- **Attacks only get better**: MEGA's key recovery went from 512 logins to 6 and then 2 within a year, so "not a realistic attack" is a weak defence

Guest Lecture: Diego F. Aranha

- In his systems security course at Aarhus, master's students analyse real apps that hold sensitive data, such as banking, health, transport and e-government; around 180 apps so far
- The good: TLS is everywhere and mostly up to date. The bad: home-made cryptography, CBC without integrity, weak algorithms, and hard-coded API keys
- The worst findings are not cryptographic: trusting the client, public storage buckets, private data leaking through APIs, and privilege escalation
- Findings are disclosed to the companies; a law against unauthorised access that does not require malicious intent makes good-faith research legally uncertain, so scope, rules of engagement and coordinated disclosure matter

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

Randomness

- Security is never better than the min-entropy of the key; aim for 128 bits
- Seed from a true random source, then use a cryptographic PRNG, never a built-in one; the dangerous moments are first boot, cloned machines and developer-chosen seeds
- A repeated ECDSA nonce gives the key by subtraction, a biased one by lattice reduction
- Deterministic nonces remove the RNG from signing; hedged nonces also survive faults
- Miller–Rabin with fixed bases can be fooled, and shared factors across keys fall to a GCD

Randomness Failures in the Wild

Failure	Example	Lesson
Too little entropy	Debian OpenSSL (2006–2008): one line removed, 2^{15} possible keys	Test the source, not only the output
Entropy at first boot	Embedded devices sharing RSA primes; Mining your Ps and Qs (2012)	Do not generate keys before the pool is seeded
Constant nonce	Sony PS3 (2010): the same ECDSA nonce for every signature	One reuse reveals the key
Broken platform RNG	Android `SecureRandom` (2013): Bitcoin wallets emptied	Use the operating system: `getrandom`, `secrets`, `OsRng`
Biased nonce	Minerva, TPM-FAIL, LadderLeak: a few leaked bits per signature	Lattices turn small biases into keys

Legacy Crypto and TLS

- MD5, SHA-1, RC4 and 3DES are broken or deprecated and are still deployed; know the attacks: chosen-prefix collisions, keystream biases, Sweet32
- Export-grade parameters survive in configurations: FREAK, Logjam and shared 512-bit groups
- Downgrade attacks target the negotiation, not the ciphers; TLS 1.3 removed the options
- Dual EC is the textbook backdoor: whoever chose the constants predicts the output; demand nothing-up-my-sleeve parameters
- Use ephemeral elliptic-curve Diffie–Hellman, AEAD and TLS 1.3, and nothing you have to configure yourself

Legacy Crypto: The Attacks

Primitive	Attack	Lesson
MD5, SHA-1	Collisions: chosen-prefix for a rogue CA (2008) and Flame (2012); the first SHA-1 collision in 2017	Collision resistance fails long before preimages do
RC4	Keystream biases and related IVs: WEP broken in minutes	A stream cipher needs a fresh, independent key per message
3DES	Sweet32 (2016): a 64-bit block repeats after about 2^{32} blocks	Block size is a security parameter too
Export RSA and DH	FREAK and Logjam: downgrade to 512 bits; one prime served 18% of HTTPS	Remove options instead of configuring them
Dual EC	Backdoored constants; exploited in Juniper firewalls (2015)	Demand nothing-up-my-sleeve parameters

How Big Must a Key Be?

Problem	Best known attack	About 128-bit security
Factoring	Number field sieve, sub-exponential; the record is RSA-250 (829 bits, 2020), while 512 bits costs hours and about $75	3072-bit modulus
Discrete log in $\mathbb{F}_p^*$	The number field sieve again; a 256-bit field fell in minutes in Moscow	3072-bit prime
Discrete log on a curve	Pollard's rho, the square root of the group order	256-bit curve
Symmetric key	Brute force; Grover is far from practical	128-bit key
Hash collision	Birthday attack, $2^{n/2}$	256-bit output

The same 256 bits give 128-bit security on a curve and almost none in a finite field: the group decides, not the key size.

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

Padding Oracles

- An oracle is any observable difference between failure modes: an error, a timing, a retransmission
- CBC padding oracles decrypt a block in $16 \cdot 256$ adaptive queries; Lucky 13 turned a MAC-then-encrypt timing difference into one
- Bleichenbacher's attack on RSA-PKCS#1 v1.5 needs a million queries and reappears every few years (DROWN, ROBOT)
- CBC is IND-CPA, not IND-CCA; use an AEAD mode, and RSA-OAEP or, better, no RSA encryption at all
- Hash functions are not MACs (length extension); use HMAC or SHA-3

Oracles Everywhere

Attack	Oracle	Fix
Vaudenay (2002)	Padding error versus MAC error in CBC	Encrypt-then-MAC, or an AEAD
Lucky 13 (2013)	MAC timing over a padding-dependent length	Constant-time record processing
POODLE (2014)	SSL 3.0 padding that is not checked at all	Remove the protocol version
Bleichenbacher (1998)	PKCS#1 v1.5 conformance of an RSA plaintext	RSA-OAEP, or no RSA encryption
DROWN, ROBOT	Bleichenbacher in SSLv2 and in modern TLS stacks	Never share keys across protocol versions
Manger (2001)	The first byte of an OAEP decoding	Identical failures, in content and in time

Every time, decryption reveals something before it knows the ciphertext is authentic.

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

Side Channels: The Setting

- Kerckhoffs's principle covers the algorithm; the implementation leaks through time, power, electromagnetic radiation, caches, sound and light
- Taxonomy: remote vs physical, software vs hardware, passive vs active, non-invasive vs invasive
- Simple power analysis reads one trace; differential and correlation power analysis average over many, using a leakage model such as the Hamming weight
- Faults are the active side: a glitch skips an instruction, and a wrong CRT signature factors N by one GCD
- Microarchitectural attacks need no probe: Flush+Reload and Prime+Probe read caches, Hertzbleed reads power through frequency, remotely

From Traces to Keys

- **Capture**: trigger on the operation, sample with the target clock, record many traces with known inputs
- **Align**: a delay before the operation is undone by the sum of absolute differences (SAD) to a reference window; delays inside it need dynamic time warping
- **Predict**: for every key guess, compute an intermediate value, such as the S-box output, and its Hamming weight
- **Correlate**: the right guess gives the highest correlation ρ between predicted and measured power; the traces needed grow like $1/\rho^2$, so countermeasures shrink ρ
- **Glitch**: a fault in the clock or the voltage skips a check or corrupts a computation, and the corrupted output is often enough

Side Channels: Attacks and Defences

- AES: correlation power analysis on the S-box output recovers the key byte by byte; realign jittery traces first
- RSA and ECC: the secret selects the operations; use regular algorithms, blinding, and constant-time arithmetic
- Validate inputs: invalid-curve and twist attacks turn a static key into small discrete logarithms
- Post-quantum schemes fall to the same attacks: power analysis on the NTT, KyberSlash, plaintext-checking oracles, leaked masks
- Masking: linear operations are cheap, non-linear ones expensive, so ML-KEM and ML-DSA cost more to protect than AES; constant time is a property of the binary

Countermeasures and Their Cost

Defence	Stops	Cost and caveats
Constant time	Timing, cache and simple power analysis	No secret branches, addresses or variable-latency instructions; check the binary
Blinding	Correlation on RSA and ECC	Randomise base, exponent or projective coordinates; cheap
Masking	Correlation of order d	$d + 1$ shares, cost about d^2; first-order AES is about $43\times$ slower
Regular algorithms	Simple power analysis	The Montgomery ladder; does nothing against faults
Verify the result	Many fault attacks	Check before release; fails for faults that still give valid outputs
Testing	Regressions	dudect and friends find leaks, they do not prove absence

Designing the Problem Away

- **Two philosophies:** validate every input and protect every operation, or choose a design in which the dangerous inputs and branches do not exist
- **Clamping** in X25519: the scalar is a multiple of the cofactor 8 with a fixed top bit, so small-order points contribute nothing and the ladder always has the same length
- **Complete formulas** in Ed25519: one addition formula is correct for all inputs, so there are no special cases and no branches
- **A friendly prime**, $2^{255} - 19$: reduction is a fixed sequence of operations, with no data-dependent path
- **Deterministic nonces** in Ed25519: no random number generator at signing time, with hedging as protection against faults

Testing and Certification

- **dudect** compares timing distributions for two classes of inputs; **ctgrind** and **TIMECOP** mark secrets as undefined memory and let Valgrind flag every branch or address that depends on them
- These tools find leaks; they cannot prove there are none, and they only see the binary and the CPU they run on
- **FIPS 140-3** has four levels for cryptographic modules, from approved algorithms with no physical protection up to tamper detection, response and hardened enclosures; only levels 3 and 4 require resistance to non-invasive attacks such as power analysis
- A certificate says that a laboratory tested one module, in one configuration, against one list of attacks; it says nothing about the attack published next year

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

Quantum-Safe Cryptography

- Shor breaks factoring and discrete logarithms; Grover's square-root speed-up is far from practical, so AES-128 and SHA-256 can stay for now
- Harvest now, decrypt later: if data must stay secret for x years and migrating takes y, a quantum computer within $x + y$ years is already a problem (Mosca)
- Lattices: LWE hides a secret behind small noise, SIS asks for a short solution; the modulus-to-noise ratio decides the hardness
- ML-KEM is ElGamal over a lattice plus the Fujisaki–Okamoto transform; ML-DSA is Schnorr with aborts, so the response hides the secret
- FN-DSA trades size for a floating-point sampler, SLH-DSA for a hash-only assumption; NIST deprecates RSA and ECC from 2030; deploy hybrid now

Post-Quantum Cryptography in Practice

Scheme	Public key	Ciphertext or signature	Use
ML-KEM-768	1 184 B	1 088 B	Key exchange, hybrid with X25519 in TLS
ML-DSA-65	1 952 B	3 309 B	Default signature
FN-DSA-512	897 B	about 666 B	Small signatures, hard to implement
SLH-DSA-128s	32 B	7 856 B	Conservative, hash-based only
X25519, Ed25519	32 B	32 B, 64 B	What we replace

Deployed today: hybrid key exchange in TLS, PQXDH in Signal, PQ3 in iMessage. Signatures and certificates are the harder part: every byte goes into every handshake.

Side Channels on Post-Quantum Schemes

Target	What leaks
ML-KEM decapsulation	The NTT and the message decoding leak to correlation power analysis; a non-constant-time re-encryption check is a plaintext-checking oracle
KyberSlash (2024)	A division by the modulus that takes secret-dependent time on many CPUs
ML-DSA signing	One leaked mask y reveals s_1; a fault on deterministic signing does the same
Masking	Linear operations are cheap to mask, but compression, sampling and the rejection step are not, so protected ML-KEM and ML-DSA cost much more than protected AES
The answer	Hedged signing, constant-time code verified on the binary, masking designed in from the start (Raccoon)

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

Crypto API Failures

- The client is malicious and every input is an attack surface; the proof assumed honest inputs, so the API must enforce them
- Rogue keys: a key chosen as a function of someone else's breaks two-party key generation and BLS aggregation; commit first, or prove possession
- Deterministic nonces protect against bad randomness, not against a co-signer who changes the challenge
- Batch verification needs random coefficients, or errors cancel
- Validate parameters, inputs and outputs; separate key roles; make failure modes identical; bind nonces to the session
- Each call may be fine on its own; the attacks live in the combinations

What the API Must Check

Check	Failure without it	Example
Signature ranges	$(r, s) = (0, 0)$ verifies every message	Java ECDSA, CVE-2022-21449 (2022)
Key roles	A wrapping key that also decrypts exports every key	PKCS#11 wrap-then-decrypt
Proof of possession	A rogue key cancels the honest keys in an aggregate	BLS aggregation
Random coefficients	Invalid signatures cancel in a batch	Batch verification of Schnorr
Identical failures	Error messages and timings become oracles	Padding oracles, again

The common thread: the specification assumed honest inputs, and the implementation trusted that assumption.

Commitments and Zero-Knowledge

- Commitments are hiding and binding; at most one property can be unconditional
- A known $\log_g h$ is a trapdoor; derive parameters from public data
- Zero-knowledge proofs are sound and zero-knowledge: the extractor and the simulator are the same computation in two roles
- The Schnorr protocol is the template and extends to many relations: equal discrete logarithms, representations and linear relations between committed values, AND by sharing the challenge, OR by splitting it
- Fiat–Shamir must hash everything the verifier sees, or the prover chooses the statement after the challenge (Helios, modern proof systems)
- SNARKs inherit all of this and add under-constrained circuits

Proof Systems in Practice

- **Where they are used:** signatures, verifiable electronic voting, anonymous credentials such as age proofs in the European identity wallet, and blockchains
- **Weak Fiat–Shamir:** 36 weak implementations across 12 proof systems (Dao et al., 2023); in one blockchain it would have allowed unlimited new currency
- **Under-constrained circuits:** the most common class among 141 catalogued SNARK vulnerabilities (Chaliasos et al., 2024): the proof is correct, the statement is wrong
- **Trusted setups:** a trapdoor in the parameters breaks soundness for everyone, so setups are run as multi-party ceremonies
- **Side channels:** the prover's randomness is a nonce

Protocol Composition

- Secure pieces do not make a secure system; the attacker chooses the composition
- MEGA: malleable key storage plus server-steered decryption gave an oracle on the primes: 1023, then 512, then 6, then 2 logins
- Telegram: a custom channel protocol needs custom proofs and custom constant-time code
- OpenPGP: configurations that are safe alone are unsafe across implementations; Moscow: three small field keys are three easy problems
- Threema: one key in two protocols lets a text message forge a credential
- One key per purpose, authenticate everything, use standard compositions, prove the whole

Five Systems, One Pattern

System	What went wrong	What fixes it
MEGA	Unauthenticated key storage, and one RSA key for sharing and login	Authenticated encryption of every key, and one key per purpose
Telegram	A home-made channel with unauthenticated metadata	A standard channel such as TLS, analysed as a whole
OpenPGP	Group and exponent sizes left to each implementation	Fix the parameters in the standard; RFC 9580 deprecated ElGamal
Moscow e-voting	Three small keys, and messages outside the subgroup	One large standard group and a correct message encoding
Threema	One long-term key in two protocols, no proof of possession	Key separation and a new protocol (Ibex)

Contents

Practical Information

The Big Picture

Randomness and Legacy Crypto

Padding Oracles

Side-Channel Attacks

Quantum-Safe Cryptography

Protocols and Proofs

Closing

A Checklist for Your Own Systems

Before you ship	Ask
Randomness	Where does every key and nonce come from, and what happens at first boot?
Primitives	Is anything on the list of the deprecated, and is anything configured by hand?
Error handling	Do all failures look, and take, exactly the same?
Implementation	Is the binary constant time on secret data, and what does a fault do?
Interfaces	Which inputs can the other side choose, and are they all validated?
Composition	Is any key used for two purposes, and who proved the whole system?
Migration	Can every primitive be replaced, and what is the plan for 2030?

Discussion

1. Which attack in this course surprised you most, and why did we not see it coming?
2. Which class of mistake from the course is most dangerous in practice today, and why?
3. Several systems had security proofs and were still broken. Are proofs worth the effort, and what should they cover?
4. If you could change one thing about how cryptography is deployed in practice, what would it be?
5. What is the biggest risk in the migration to post-quantum cryptography: the schemes, the implementations, or the migration itself?
6. Which topic would you have liked to spend more time on, and which less?

Where to Go From Here

- **Follow the attacks:** the talks, podcasts and blog posts linked to each lecture in the lecture plan on the course website, the IACR ePrint archive, and Real World Crypto
- **Build and break:** CryptoHack, the ChipWhisperer community, and the open-source libraries whose bugs we studied
- **Go deeper:** master's projects on side channels against post-quantum cryptography, on digital identity wallets, and on lattice-based signatures are available at our group
- **Stay humble:** every system in this course was built by (mostly) competent people who believed it was secure

Read, Watch and Listen

Every lecture in the lecture plan on the course website links to talks, podcasts and blog posts on its topic. Some favourites:

Topic	Start with
Crypto wars	*Keys Under Doormats* (Rivest); *Crypto Wars* (Darknet Diaries)
Randomness	*Reality of Entropy* (Valsorda); *Biased Nonce Sense* (Heninger)
Legacy and TLS	*Logjam* (Halderman, Heninger); Dual EC with Matthew Green (podcast)
Padding oracles	Bleichenbacher's attack (Wong, video); *ROBOT* (USENIX 2018)
Side channels	Paul Kocher's keynote (USENIX 2018); Colin O'Flynn on ChipWhisperer
Post-quantum	*Kyber* (Schwabe); high-assurance Kyber (podcast)
Protocols	*Weak Fiat–Shamir*; *Breaking MEGA*; *Lessons From Threema*

Beyond the course: Matthew Green's blog and the podcast *Security Cryptography Whatever*.

Final Thoughts

From what I can see, you have learned a lot and performed well this semester. I am certain that the way of thinking, our discussions, and the problems you have solved in this course will be useful for all of you going forward.

I hope that you enjoyed the course, that it was challenging but interesting, and that you see the value of your effort. Good luck with the assignments!