NTNU | Norwegian University of Science and Technology

TTM4205 – Technical Essay, Fall 2026

Secure Cryptographic Implementations

Assignment

This is one out of three assignments in the course TTM4205 Secure Cryptographic Implementations during the fall semester of 2026. In this assignment you write a *technical essay* and give a *presentation* about a scientific topic related to the course, either one not covered by the lectures or one covered in more depth. This is joint work in *groups of 2 to 3 students*, both the essay and the slides must be written in LaTeX, and the topic, scope, and group must be *approved* by the course staff.

The assignment counts for at most 30 points, along three criteria worth 10 points each: technical content and significance; analysis and reasoning; and structure, clarity, and referencing. The presentation carries no points of its own, but it is *mandatory*: if your group does not present, your essay is not graded. Contact us if illness or another valid reason prevents your group from presenting.

Length. The essay must be 8 *to* 10 *pages*, and we subtract 1 point per page outside that range. The first page (title, authors, emails, abstract), the references, and an optional appendix do *not* count; use the appendix for supporting material that does not fit in the main body.

Use of AI. What you hand in must be your own work: the analysis, the explanations, and the text must come from you. You are *not* allowed to use AI tools to solve the assignment or to produce your write-up. You *may* use AI tools to polish a finished submission, for example to tidy up code or improve wording and formatting. In this assignment, you may also use AI to research your topic and to review a draft, and you must state in a *footnote on the first page of the essay* how, if at all, you used it.

Templates and resources. Mandatory LaTeX templates: overleaf.com/read/nhcnrbnwzmcw (essay) and overleaf.com/read/zjqxggmjnzqp (presentation). See also the NTNU rules on cheating and academic writing, the NTNU English style guide, Simon Peyton Jones on writing a great research paper, and Cryptobib. If you get stuck, ask on the Canvas forum or talk to us.

Deadlines. Topic, scope, and group approval by October 30th; presentations on November 10th, 13th, or 20th; final submission by December 18th at 23:59. If you pick from our list of topics, your proposal must name at least *two* of them in order of preference, so that we can spread the groups out; your own topic needs only one suggestion, with a title, scope, and two references. Rank the three presentation dates as well, since we assign slots from those rankings.

Submission. Submit your work in Canvas; the submission link is available from the course website at ttm4205.iik.ntnu.no.

Suggested Topics

The topics below are ready to use: each one states what to write about and gives you references to start from, so you can pick one as it stands and begin working. Narrowing the scope further is welcome but not required, and we allow at most two groups on the same topic.

You are equally welcome to propose a topic of your own. An essay on something you already find interesting is usually the better essay, and anything about the design, implementation, analysis, or deployment of cryptography in practice is fair game. For your own topic, provide a (preliminary) title and scope together with at least two references. Ask the course staff early if you are unsure whether an idea works.

Cryptographic Fuzzing

It is hard to verify whether a given piece of cryptographic code is securely implemented. One way to find bugs is *fuzzing*: automatically feeding malformed or random inputs to a target and watching for crashes, incorrect outputs, or leaks. Cryptography-specific fuzzers can find protocol-state bugs, memory-safety issues, and differences between implementations that should behave identically [Som16]; see, e.g., the Cryptographic Fuzzing framework at github.com/kudelskisecurity/cdf. A complementary approach is to learn the state machine of an implementation by black-box testing and compare it against the specification, which revealed flaws in GnuTLS, OpenSSL, and Java Secure Socket Extension [dRP15]. Discuss what kinds of vulnerabilities fuzzing is good at finding, and how to design good oracles for cryptographic targets.

Static Analysis of Cryptographic Code

In contrast to fuzzing, *static analysis* inspects code without running it. It can discover the wrong usage of randomness, deprecated or weak algorithms, hard-coded keys, or missing API checks [LJL+22]. The approach scales: an early study of 11 748 Android applications found that 88% of those using cryptography made at least one mistake against a short list of simple rules [EBFK13]. Discuss which classes of vulnerability static analysis can and cannot catch, the problem of false positives, and how it complements dynamic techniques such as fuzzing.

Formally Verified Cryptographic Implementations

While fuzzing and static analysis are reactive solutions that require work after the code is written, a more proactive approach is to only allow correct and secure code to be written in the first place. Projects such as HACL* [ZBPB17], Fiat-Crypto [EPG+20], and Jasmin [ABB+17] produce machine-checked, high-speed implementations together with proofs of functional correctness and, in some cases, constant-time behavior. Microsoft's Project Everest at microsoft.com/en-us/research/blog/project-everest discusses what it takes to deploy such code.

Machine-Checked Cryptographic Proofs

Beyond the implementations, one can also machine-check the *security proofs* of the schemes themselves. EasyCrypt [BGHZB11] is a proof assistant for game-based cryptographic proofs, while CryptoVerif [Bla08] works in the computational model and can produce proofs of protocols largely automatically. Discuss what *provable security* guarantees when the proof is machine-checked rather than written by hand, and how EasyCrypt connects to verified implementations through the joint Jasmin/EasyCrypt toolchain.

Cryptographic Safety in Rust

Memory-safety bugs (such as Heartbleed) and timing leaks are two of the most common failure modes in cryptographic software. Rust's ownership model rules out many memory-safety bugs by construction, and the ecosystem provides constant-time building blocks such as the `subtle` crate and the RustCrypto libraries (github.com/RustCrypto). Discuss what Rust does and does not protect against: it does not by itself guarantee constant-time execution, so implementations still rely on careful coding and on tools such as `dudect` [RBV17] and constant-time verification [ABB+16].

Choosing Cryptographic Parameters

How do we pick key sizes, security levels, and round counts for real deployments? Discuss the relationship between the assumed hardness of the underlying mathematical problem (factoring, discrete logarithms, or Learning With Errors) and the recommended parameters, the role of security margins, and standard recommendations such as NIST SP 800-57 and keylength.com. A good starting point is Aumasson's essay "Too Much Crypto" [Aum19], which argues that many primitives use larger security margins than necessary. For the post-quantum side, Albrecht et al. [ACD+18] show how lattice parameters are actually estimated, and how much the estimates depend on the cost model chosen for the underlying attack.

Vulnerabilities in Threshold Signatures

Lindell published a threshold signature scheme [Lin21] based on the Elliptic Curve Digital Signature Algorithm (ECDSA). This scheme was later implemented and used in practice, and while the construction was theoretically secure, the implementations contained bugs that allowed an attacker to extract the secret key [AS20]. See also the report at fireblocks.com/blog/lindell17-abort-vulnerability-technical-report. There are more recent attacks on threshold ECDSA by Makriyannis, Yomtov, and Galansky [MYG23] on similar schemes, presented at a NIST workshop: csrc.nist.gov/presentations/2023/mpts2023-day2-talk-attack-threshold-ecdsa-wallets.

Degenerate Edwards Curve Attacks

We get into trouble if we do not verify that points $P = (x, y)$ are on the (Weierstrass) elliptic curve $E(\mathbb{F}_p) : y^2 = x^3 + a \cdot x + b$ [ABM+02]. Furthermore, the addition formulas are not complete, which means that the way we compute the addition of two points P and Q on $E(\mathbb{F}_p)$ depends on the input. This makes implementations more complicated to get correct and enforces complex side-channel countermeasures, since the difference in addition method may leak secret key data.

Edwards curves $Ed(\mathbb{F}_p) : y^2 - x^2 = 1 + d \cdot x^2 \cdot y^2$ (simplified) were introduced to solve these issues, leading to the more efficient and secure signature scheme EdDSA [BDL+11], later standardized by the IRTF at datatracker.ietf.org/doc/html/rfc8032 (also including a Python implementation). However, these curves are also vulnerable to specially crafted input points, as shown by Neves and Tibouchi [NT16].

Randomness Failures in the Wild

Many real-world breaks come not from broken mathematics but from broken randomness. Examples include widely shared RSA prime factors caused by low-entropy key generation [HDWH12], the ROCA vulnerability in Infineon smart cards [NSv+17], and the Debian OpenSSL disaster, where a two-line patch reduced the entropy of every key generated over almost two years to the process ID [YRS+09]. Survey a few such incidents, explain the underlying entropy failures, and draw lessons for randomness sources and key generation.

Padding-Oracle Attacks in Practice

Bleichenbacher's 1998 attack on RSA PKCS#1 v1.5 keeps coming back. The ROBOT work [BSY18] showed that many TLS stacks were still vulnerable two decades later, and the Lucky 13 attack exploited CBC padding in TLS through timing [AP13]. The Marvin attack [Kar24] then showed that the timing variant is far from dead: using statistical tests on decryption timings alone, it broke OpenSSL, GnuTLS, NSS, and others, and argues that the depadding code cannot realistically be made constant time. Survey how these attacks work, why they persist despite being so well understood, and which countermeasures are deployed today.

Side-Channel Attacks Against Post-Quantum Cryptography

The schemes NIST has already standardized are well studied: see, e.g., [MGTF19] for an analysis of rejection sampling, number-theoretic transforms, and polynomial multiplications in Dilithium. The ongoing process is more interesting. NIST runs a separate call for *additional* signature schemes, and nine candidates advanced to its third round in May 2026, a round that explicitly puts more weight on implementation security and resistance to physical attacks; see csrc.nist.gov/projects/pqc-dig-sig/round-3-additional-signatures. The lattice-based candidate HAWK was withdrawn shortly afterwards, following a key-recovery attack that cut its security margin roughly in half, leaving eight candidates on quite different assumptions.

Three of them make good essays. **MAYO** is a compact UOV-style multivariate scheme, and single-trace power analysis of its modular multiplication already recovers the secret key on a Cortex-M4 [JD24]. **FAEST** is built from AES and VOLE-in-the-head rather than a number-theoretic assumption, and even a masked implementation falls to side-channel and fault attacks on the VOLE generation and the all-but-one vector commitments [JD25]. **SDitH** is the one code-based candidate left in the round, an MPC-in-the-head scheme over syndrome decoding, and a soft-analytical attack on the leakage of a single Galois-field multiplication recovers its secret from one trace [GAG+25]. Pick one, explain where its secret-dependent operations are, and analyse what an attacker with power or timing traces could learn.

Microarchitectural Side Channels

Side channels are not limited to hardware you can attach a probe to. On ordinary server and desktop CPUs, shared caches let one process observe the memory-access pattern of another: FLUSH+RELOAD recovers RSA private keys from GnuPG across processes, and even across virtual machines on the same host [YF14]. Hertzbleed goes further and turns a power side channel into a purely remote timing attack, because dynamic frequency scaling makes the CPU clock depend on the data being processed, which broke a constant-time implementation of SIKE [WPH+22]. Discuss what *constant time* can and cannot mean on such a platform, and compare these attacks with physical measurements on an embedded target.

Backdoors and Kleptography

A cryptographic standard can also fail because someone wanted it to. The Dual EC DRBG is the textbook case: whoever chooses the two curve points can hold a trapdoor that recovers the generator's internal state, and thus predict all future output. Checkoway et al. measured how exploitable this actually was in TLS implementations [CFN+14], and later analysed the Juniper ScreenOS incident, where unauthorized code changed the Dual EC point in a shipping VPN product and enabled passive decryption [CMG+16]. Matthew Green's "The Many Flaws of Dual_EC_DRBG" at blog.cryptographyengineering.com is a very readable starting point. Discuss how a backdoor can be hidden in plain sight in a standard, what makes a design *nothing up my sleeve*, and which properties let outsiders detect such sabotage.

Nonce-Misuse-Resistant Authenticated Encryption

Many real-world failures of authenticated encryption come from nonce reuse: reusing a nonce with AES-GCM leaks the authentication key and breaks integrity (the *forbidden attack*) [BZD+16]. Nonce-misuse-resistant schemes such as AES-GCM-SIV [GL15] are designed to degrade gracefully when nonces repeat. Discuss why nonce reuse is so dangerous, how misuse-resistant AEAD works, and the trade-offs it makes.

Migrating TLS to Post-Quantum Cryptography

As post-quantum algorithms are standardized, protocols such as TLS must migrate without waiting for the threat to materialize. The common approach is *hybrid* key exchange that combines a classical scheme (e.g., X25519) with a post-quantum KEM (e.g., ML-KEM), so security holds if either component is secure. Discuss the design and performance trade-offs of post-quantum and hybrid TLS [SKD20], including handshake size and latency. NIST's draft transition roadmap [MPR+24] sets the wider timeline, proposing that the classical public-key algorithms be deprecated from 2030 and disallowed from 2035; a good essay takes *harvest now, decrypt later* seriously and asks which parts of a protocol actually need to migrate first.

Cryptographic Agility

The migration above raises a more general question: how do you build a system in which an algorithm can be replaced at all? Most deployed software hard-codes its cryptography in ways that make substitution expensive — key and signature sizes baked into formats and databases, algorithm identifiers that cannot be negotiated, certificates and protocol versions that assume one specific primitive. *Cryptographic agility* is the discipline of designing so that a primitive can be swapped without redesigning the system, and it is what determines whether the post-quantum transition takes months or a decade.

The workshop report by Ott, Peikert, et al. [OP+19] maps out the research problems, a systematic review by Alnahawi et al. [ASW+23] shows that the community does not even agree on a definition, and NIST's white paper [Nat25] collects the concrete mechanisms and their trade-offs. Discuss what agility costs: negotiation adds complexity and attack surface, as the downgrade attacks on TLS demonstrate, so more agility is not automatically more security.

Cryptography in the Go Programming Language

Go is an interesting case study because its cryptography lives in the standard library, is written in Go rather than C, and is maintained as one coherent design. Two developments are worth an essay. Go now ships a native *FIPS 140-3* module — the same standard-library code, validated and switchable at run time — instead of the old cgo-based BoringCrypto detour, and the write-up at go.dev/blog/fips140 is candid about where compliance and security pull in opposite directions. Go has also adopted post-quantum cryptography early: `crypto/mlkem` entered the standard library in Go 1.24, and `crypto/tls` negotiates the hybrid X25519MLKEM768 by default.

Useful sources are CryptoGo [LJL+22] on misuse of Go's cryptographic APIs, the audit at go.dev/blog/tob-crypto-audit, and Arquint's machine-checked repair of a deviation in Go's extended-GCD implementation, which also made it 24% faster [Arq26]. Discuss what a memory-safe language with a curated API buys you, and what it does not: the language still does not guarantee constant-time behaviour.

Implementation Bugs in Zero-Knowledge Proof Systems

Zero-knowledge proofs are now deployed at scale, and the gap between the proven protocol and the shipped code is wide. A systematization of 141 real SNARK vulnerabilities [CET+24] shows that most of them are not in the proof system but in the circuit and the surrounding layers, with under-constrained circuits the dominant class: the circuit accepts witnesses it was never meant to accept, so soundness is lost while every test still passes. A second failure mode is applying the Fiat-Shamir transform incorrectly by hashing too little of the transcript; a survey of open-source code found 36 such weak implementations across 12 proof systems, one of which could have allowed untraceable theft [DMWG23]. Explain why these bugs are invisible to ordinary testing, and what tooling or discipline would catch them.

Verified Constant-Time and Secure Compilation

Even a constant-time source program can leak through timing if the compiler optimizes away the countermeasures. Domain-specific languages and verified toolchains such as FaCT [CSJ+19] aim to preserve constant-time behavior down to the compiled code, and the CompCert-based compiler of Barthe et al. [BBG+20] goes further by *proving* that every constant-time source program is compiled to constant-time machine code. Discuss why *constant-time in the source* is not enough, and how secure compilation and verification address this.

References

[ABB+16] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *USENIX Security Symposium*, 2016. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida

[ABB+17] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. Jasmin: High-assurance and high-speed cryptography. In *ACM Conference on Computer and Communications Security (CCS)*, 2017. https://doi.org/10.1145/3133956.3134078

[ABM+02] Adrian Antipa, Daniel Brown, Alfred Menezes, René Struik, and Scott Vanstone. Validation of elliptic curve public keys. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-36288-6_16

[ACD+18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In *Security and Cryptography for Networks (SCN)*, 2018. https://doi.org/10.1007/978-3-319-98113-0_19

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy (S&P)*, 2013. https://doi.org/10.1109/SP.2013.42

[Arq26] Linard Arquint. GCD: Garbled, corrected, demonstrandum – fixing and proving Go's extended GCD implementation. arXiv:2606.05796, 2026. https://doi.org/10.48550/arXiv.2606.05796

[AS20] Jean-Philippe Aumasson and Omer Shlomovits. Attacking threshold wallets. Cryptology ePrint Archive, Paper 2020/1052, 2020. https://eprint.iacr.org/2020/1052

[ASW+23] Nouri Alnahawi, Nicolai Schmitt, Alexander Wiesmaier, Andreas Heinemann, and Tobias Graßmeyer. On the state of crypto-agility. Cryptology ePrint Archive, Paper 2023/487, 2023. https://eprint.iacr.org/2023/487

[Aum19] Jean-Philippe Aumasson. Too much crypto. Cryptology ePrint Archive, Paper 2019/1492, 2019. https://eprint.iacr.org/2019/1492

[BBG+20] Gilles Barthe, Sandrine Blazy, Benjamin Grégoire, Rémi Hutin, Vincent Laporte, David Pichardie, and Alix Trieu. Formal verification of a constant-time preserving C compiler. *Proceedings of the ACM on Programming Languages (POPL)*, 4, 2020. https://doi.org/10.1145/3371075

[BDL+11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23951-9_9

[BGHZB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO*, 2011. https://doi.org/10.1007/978-3-642-22792-9_5

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4), 2008. https://doi.org/10.1109/TDSC.2007.1005

[BSY18] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of Bleichenbacher's oracle threat (ROBOT). In *USENIX Security Symposium*, 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[BZD+16] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2016. https://www.usenix.org/conference/woot16/workshop-program/presentation/bock

[CET+24] Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. SoK: What don't we know? understanding security vulnerabilities in SNARKs. In *USENIX Security Symposium*, 2024. https://www.usenix.org/conference/usenixsecurity24/presentation/chaliasos

[CFN+14] Stephen Checkoway, Matthew Fredrikson, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, and Hovav Shacham. On the practical exploitability of Dual EC in TLS implementations. In *USENIX Security Symposium*, 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/checkoway

[CMG+16] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the Juniper Dual EC incident. In *ACM Conference on Computer and Communications Security (CCS)*, 2016. https://doi.org/10.1145/2976749.2978395

[CSJ+19] Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. FaCT: A DSL for timing-sensitive computation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019. https://doi.org/10.1145/3314221.3314605

[DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak Fiat-Shamir attacks on modern proof systems. In *IEEE Symposium on Security and Privacy (S&P)*, 2023. https://doi.org/10.1109/SP46215.2023.10179408

[dRP15] Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In *USENIX Security Symposium*, 2015. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter

[EBFK13] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in Android applications. In *ACM Conference on Computer and Communications Security (CCS)*, 2013. https://doi.org/10.1145/2508859.2516693

[EPG+20] Andres Erbsen, Jade Philipoom, Jason Gross, Robert Sloan, and Adam Chlipala. Simple high-level code for cryptographic arithmetic: With proofs, without compromises. *SIGOPS Oper. Syst. Rev.*, 54(1), aug 2020. https://doi.org/10.1145/3421473.3421477

[GAG+25] Julie Godard, Nicolas Aragon, Philippe Gaborit, Antoine Loiseau, and Julien Maillard. Single trace side-channel attack on the MPC-in-the-Head framework. In

Post-Quantum Cryptography (PQCrypto), 2025. https://doi.org/10.1007/978-3-031-86602-9_10

[GL15] Shay Gueron and Yehuda Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In *ACM Conference on Computer and Communications Security (CCS)*, 2015. https://doi.org/10.1145/2810103.2813613

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, 2012. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger

[JD24] Sönke Jendral and Elena Dubrova. Single-trace side-channel attacks on MAYO exploiting leaky modular multiplication. Cryptology ePrint Archive, Paper 2024/1850, 2024. https://eprint.iacr.org/2024/1850

[JD25] Sönke Jendral and Elena Dubrova. Side-channel and fault injection attacks on VOLEitH signature schemes: A case study of masked FAEST. Cryptology ePrint Archive, Paper 2025/378, 2025. https://eprint.iacr.org/2025/378

[Kar24] Hubert Kario. Everlasting ROBOT: The Marvin attack. In *European Symposium on Research in Computer Security (ESORICS)*, 2024. https://doi.org/10.1007/978-3-031-51479-1_13

[Lin21] Yehuda Lindell. Fast secure two-party ecdsa signing. *J. Cryptol.*, 34(4), oct 2021. https://doi.org/10.1007/s00145-021-09409-9

[LJL+22] Wenqing Li, Shijie Jia, Limin Liu, Fangyu Zheng, Yuan Ma, and Jingqiang Lin. Cryptogo: Automatic detection of go cryptographic api misuses. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, New York, NY, USA, 2022. Association for Computing Machinery. https://doi.org/10.1145/3564625.3567989

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking dilithium. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, Cham, 2019. Springer International Publishing. https://doi.org/10.1007/978-3-030-21568-2_17

[MPR+24] Dustin Moody, Ray Perlner, Andrew Regenscheid, Angela Robinson, and David Cooper. Transition to post-quantum cryptography standards. NIST IR 8547 (initial public draft), 2024. https://doi.org/10.6028/NIST.IR.8547.ipd

[MYG23] Nikolaos Makriyannis, Oren Yomtov, and Arik Galansky. Practical key-extraction attacks in leading MPC wallets. Cryptology ePrint Archive, Paper 2023/1234, 2023. https://eprint.iacr.org/2023/1234

[Nat25] National Institute of Standards and Technology. Considerations for achieving crypto agility: Strategies and practices. NIST Cybersecurity White Paper 39, 2025. https://doi.org/10.6028/NIST.CSWP.39

[NSv+17] Matúš Nemec, Marek Sýs, Petr Švenda, Dušan Klinec, and Vashek Matyáš. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In *ACM Conference on Computer and Communications Security (CCS)*, 2017. https://doi.org/10.1145/3133956.3133969

[NT16] Samuel Neves and Mehdi Tibouchi. Degenerate curve attacks. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key*

Cryptography – PKC 2016, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-49387-8_2

[OP+19] David Ott, Chris Peikert, et al. Identifying research challenges in post quantum cryptography migration and cryptographic agility. Computing Community Consortium workshop report, 2019. https://doi.org/10.48550/arXiv.1909.07353

[RBV17] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *Design, Automation & Test in Europe (DATE)*, 2017. https://doi.org/10.23919/DATE.2017.7927267

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-quantum authentication in TLS 1.3: A performance study. In *Network and Distributed System Security Symposium (NDSS)*, 2020. https://doi.org/10.14722/ndss.2020.24203

[Som16] Juraj Somorovsky. Systematic fuzzing and testing of tls libraries. In *CCS '16*, New York, NY, USA, 2016. Association for Computing Machinery. https://doi.org/10.1145/2976749.2978411

[WPH+22] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In *USENIX Security Symposium*, 2022. https://www.usenix.org/conference/usenixsecurity22/presentation/wang-yingchen

[YF14] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*, 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[YRS+09] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In *ACM Internet Measurement Conference (IMC)*, 2009. https://doi.org/10.1145/1644893.1644896

[ZBPB17] Jean-Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. Hacl*: A verified modern cryptographic library. In *CCS '17*, New York, NY, USA, 2017. Association for Computing Machinery. https://doi.org/10.1145/3133956.3134043