A Hitchhiker's Guide to Cryptography Code Audit

Tommaso Gagliardoni
Marco Macchetti
Sylvain Pelissier

Presented at the NIST Crypto Reading Club, on 2024-02-24

KUDELSKI SECURITY

Who are we?

Tommaso Gagliardoni

- PhD from TU Darmstadt
- Cryptography + Quantum
- CRYPTO,EC,CCS,PQCRYPTO…
- @tomgag@infosec.exchange

Marco Macchetti

- Hardware security design
- Applied cryptography and cryptanalysis
- marco.macchetti@nagra.com

Sylvain Pelissier

- Security researcher
- Applied Cryptography
- CTF player
- @ipolit@mastodon.social

KUDELSKI SECURITY

Introduction

Philosophy

What is a cryptographic code audit? What is different from a traditional code audit?

Who can do a crypto code audit?

Who needs a crypto code audit?

What is expected? What if everything goes well? What if something goes wrong?

What is the value of a crypto code audit?

FINTECH BANKING CAPITAL MARKETS DIGITAL ASSETS SUSTAINABILITY ESG PRESS RELEASES

io.finnet and Kudelski Security Uncover Four Critical Vulnerabilities In Popular Digital Signature Protocols For MPC Wallets

Press Release March 21, 2023

Facebook Twitter

io.finnet

Security Audit

A full review of this library was carried out by Kudelski Security and thei
2019. A copy of this report audit-binance-tss-lib-final-20191018.pdf
this repository.

BINANCE

Implementing SNARKs securely is hard

- Moving complicated zero-knowledge protocols from theory to practice is hard.
- Suddenly it really *really* matters that the security proof is correct.
- As a community we are still learning the best practices for how to ensure this.

EasyCrypt	Audits	Standards Efforts
Peer Review	Waiting a While	Independent Proofs

ING

What is the cost of a crypto code audit?

Factors to consider

- Is the client from North Korea or similar?
- Do they want to pay us in [random sh*tcoin]?
- Are we up to the task?
    - Do we have the right people?
    - Do we have enough time?
    - Do we have availability?
- How much work is it?
    - Number of LoC
    - Dependencies
    - Complexity
    - Documentation
- Is a code audit really feasible/necessary?

Code audit process

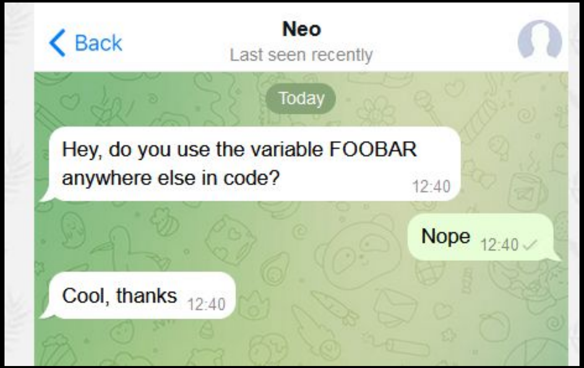
Engagement

- Client reaches out to us, typically via referral or website, contact form, etc
- Pre-sales person is assigned to the case to acquire info, sign NDA if necessary, etc
- Technical people (us) get onboard to scope the engagement
- A proposal is prepared by the Sales dept. and sent to the customer
- If accepted, a PM is assigned, a team is formed (minimum 2 auditors), and a kickoff call is scheduled

That is, in theory…

Preparation

- During kickoff, questions are asked
  - Fine-tuned schedule constraints
  - Additional documentation
  - Point of contacts
  - Threat model
- Quick communication channel with devs



The Audit

- Download code, setup private repo
- Get familiar with specs and documentation
- Ramp-up if necessary
- As a preliminary step: compile, run tests, run Automated tools if possible
- Examine line-by-line
- Make annotations, discuss internally and with client if necessary
- Create draft report, send to client for review
- Wait for client fixes, feedback, tests, etc
- If found vuln with widespread impact, prepare responsible disclosure etc
- Create final report

Aftermath

When everything is fixed or set-up:

- Publish the report if possible
- Report vulnerabilities with a CVE number
- Publish a blog post to detail the findings

What could possibly go wrong

- Documentation is in Chinese, client suggests using Google Translate
- Client sends a list of single lines to audit
- “Hey, we found a big vuln” -> Client ghosts us
- Client takes forever to fix, asks to keep confidentiality
- “Oh, we forgot to say, can we pay you in [random sh*tcoin]?”
- “Great job, thanks! In the meantime we did a couple of commits, can you start over again?”
- We miss something obvious
- We miss something important

The case of threshold crypto

Threshold signatures

MPC and threshold cryptography are quite popular targets for implementation at production level

Blockchain / secure wallets (high market value)

Companies rush in developing libraries to implement threshold signatures and more advanced schemes (e.g. hierarchical threshold)

There are different known methods in the literature

- Lindell
- GG18/GG20/CMP/CGGMP
- DKLS19/DKLS23
- Frost
- BLS
- …

Threshold signatures

No established standard so far (pick your favorite)

NIST upcoming effort on standardization (full schemes and sub-components)

Many customers choose GG18/GG20/CGGMP or FROST approaches

Resonance from crypto conferences and forum discussions

Let’s take it as example and trace for our discussion

- Paper/documentation
- Technical content
- Pain points

Papers VS Specs

Quite often, a paper is published in several versions (not all peer reviewed)

- Authors can fix things

1.2 The issue with the previous version

The previous version of the protocol uses the same "multiplicative to additive" share conversion protocol presented in [27]. As pointed out in [?, ?], the claimed simulatability property for that

[2]We point out that our work is independent from these.

Cryptology ePrint Archive

Paper 2020/540

One Round Threshold ECDSA with Identifiable Abort

Rosario Gennaro and Steven Goldfeder

Note: Second Revisions fixes issues with the multiplicative to additive share conversion protocol. First Revision fixes a typo in the malicious player identification protocol, and a typo in the evaluation graph, and a confusing sentence in the introduction.

Note: This report is now obsolete and readers should refer to the joint paper [8] which subsumes it. The paper below is a revised version of the previous eprint version which fixes some crucial details in the protocol. The proof of the protocol described in the previous version is not correct, though no attack has been shown that exploits the bug in the proof. More details appear in the Introduction.

Papers VS Specs

A paper references previous and contemporary attempts/constructions and may reuse concepts and components without describing them in detail

It is perfectly fine

An academic paper is not a specification: its goal is to present new techniques and compare them against existing ones

- Phase 3 Let $N_i = p_i q_i$ be the RSA modulus associated with E_i. Each player P_i proves in ZK that he knows x_i using Schnorr's protocol [46], that N_i is square-free using the proof of Gennaro, Micciancio, and Rabin [32], and that h_1 h_2 generate the same group modulo N_i.

Papers VS Specs

To build the chain

paper -> specification -> implementation

you have thus to follow all ramifications of a paper

A paper can be extended and merged with others

GG18 → GG20 → CGGMP

CMP → CGGMP

Authors automatically "outdate" previous papers, but for deployed implementations and libraries it is much more complex

Projects can be abandoned, forked, loosely maintained

Papers VS Specs

What is the impact of a discovered weakness/flaw?

Is it impacting a single version of a protocol or also previous/next versions? Paper version? Library version?

Customers ask for consulting, not simple collection of info

Sometimes a weakness is tagged as low importance by us, because we can't immediately produce a path to a working attack

But that doesn't mean it should not be patched

TSShock story -> Sylvain will speak about it later

Implementations

An implementation typically relies on existing libraries (no one wants to reinvent the wheel)

The panorama is varied, as implementations can be in several languages, e.g. Rust or Golang or C or Python

Each having its own ecosystem and peculiarities

Often we don't audit such dependencies, unless explicitly requested

Implementations

A threshold signature scheme is not a simple primitive, but rather a complex protocol composed by many pieces (e.g. Groth-Shoup 23 is 99 pages long!)

To implement GG20 threshold ECDSA we have to implement:

- Paillier encryption and RSA modulos -> safe primes
- Good randomness sources to sample uniformly
- Network protocols to connect parties
- Zero knowledge proofs
- Multiplicative to additive share conversion
- Commitment algorithms
- Verifiable secret sharing
- Elliptic curve cryptography

What could go wrong? Randomness

Basic assumption for any scheme

- Phase 1. Each Player P_i selects $k_i, \gamma_i \in_R Z_q$; computes $[C_i, D_i] = \mathsf{Com}(g^{\gamma_i})$ and broadcast C_i.

Uniform sampling of random values in a given range

Bad libs/PRNGs [MT, python's random]

No checks on returned randomness [error code and length]

Modulo bias from using truncated values and/or simple modulo reduction

Can lead to key compromise e.g. biased ECDSA nonces

What could go wrong? Randomness

Solution #1: rejection sampling

- Repeat sampling from wider range (typically the nearest power of 2) and discard value if not in correct range

Solution #2: sampling from a wider range and reduce with modulo

- If range is extended by 128 bits, reducing modulo q is fine. Expected bias is 2^-128, typically comparable with the scheme’s claimed security level

What could go wrong? Networking (1)

Paper assume P2P and broadcast communication protocols, without discussing their implementation

Communication between machines over a network is modeled by way of subroutine-machines that represent the behavior of the actual communication network under consideration. In this work we assume for simplicity that the parties are connected via an authenticated, synchronous broadcast channel. That is, the computation proceeds in rounds, and each message sent by any of of the parties at some round is made available to all

Broadcast is especially tricky because we have to ensure all parties receive the same messages

Can be easy if trusted dealer is present

Otherwise, implementations try to optimize by re-using P2P connections to mock broadcast

What could go wrong? Networking (1)

Example is key refreshing; parties refresh their private key shares after key generation

In case of P2P used in place of broadcast, they finally send each other ACK/NOTACK with an additional round

But a malicious player can send ACK to half parties (which will update share) and NOTACK to the other half (which will discard new share)

Key is lost! Forget and forgive attack

What could go wrong? Networking (1)

A way to fix is to introduce one more round where parties send each other the full lists of ACK/NOTACK answers from the previous round

But a malicious party can again send a full ACK list to some parties and different lists to others!

Leading to “improved-yet-another-ack” follow ups, etc…

Solution: use published solution such as echo broadcast (Goldwasser Lindell 2002)

What could go wrong? Networking (2)

P2P connections must be encrypted and authenticated

key exchange → shared symmetric key

This is fine, but papers also include techniques to identify dishonest parties (identifiable aborts)

In this case, shared keys cannot provide non-repudiability

Single phrase in GG20 paper identifiable aborts section:

First of all we assume that all messages transferred between players are signed, so that it is possible to determine their origin.

What could go wrong? Networking (3)

Paper assume that single runs of the protocol are unique and that values cannot be replayed from one execution to the next

CGGMP introduces ssid everywhere

– When obtaining output $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$ and (x_i, y_i, p_i, q_i), set $ssid = (sid, rid, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{N}, \boldsymbol{s}, \boldsymbol{t})$

– Sample $\rho_i, u_i \leftarrow \{0,1\}^\kappa$ and compute $V_i = \mathcal{H}(ssid, i, \boldsymbol{X}_i, \boldsymbol{A}_i, Y_i, B_i, N_i, s_i, t_i, \hat{\psi}_i, \rho_i, u_i)$. Broadcast $(ssid, i, V_i)$.

but this is not stated explicitly in other papers allowing replays of messages

What could go wrong? Commitments

Care has to be taken manipulating values in case of type cast and/or concatenations, language specifics

Array of compressed ECC points entering an hash to compute a commitment

$$c = H(r, P1, P2, \ldots)$$

Points are cast to bytes from int to build the hash input

Golang int.Bytes()

if #bytes is not specified during conversion, 0x00 prefix bytes are ripped off

What could go wrong? Commitments

A customer had such a function, that moreover inserted a separator ‘$’ after each point

Consider the following two pairs of points A,B (ints):

[0x00 A1 … A31] , [‘$’ B1 … B31]

[A1 … A31 ‘$’] , [0x00 B1 … B31]

When Bytes is called and ‘$’ delimiters are put, in both cases we get:

A1 … A31 ‘$’ ‘$’ B1 … B31 ‘$’

Collision! -> Sylvain will talk more about this now!

Input malleability

Hash commitments

You commit to a value *v* but do not reveal it in advance:

$$H(e|v)$$

e is a blinding value used for randomization.

Revealing *(e,v)* later, allows everyone to verify the commitment.

Hash commitment problems

There is a lack of separation between the blinding and the committed values:

$$H(0\text{x}1337|0\text{x}1000) = H(0\text{x}133710|0\text{x}00)$$

We have the same commitment for two different values. The scheme is not binding.

Commitment example

```
export namespace HashCommitment {

export function createComWithBlind (message: BN, blindFactor: BN): BN {
  const sha256 = cryptoJS.algo.SHA256.create()
  sha256.update(Hex.toCryptoJSBytes(Hex.padEven(blindFactor.toString(16))))
  sha256.update(Hex.toCryptoJSBytes(Hex.padEven(message.toString(16))))
  const dig = sha256.finalize()
  return new BN(cryptoJS.enc.Hex.stringify(dig), 16)
}
```

Commitment example

```
var msg1 = new BN("1000", 16)
var blind1 = new BN("1337", 16)
var com1 = Cls.createComWithBlind(msg1, blind1)
console.log(com1.toString(16))

var msg2 = new BN("00", 16)
var blind2 = new BN("133710", 16)
var com2 = Cls.createComWithBlind(msg2, blind2)
console.log(com2.toString(16))

assert.strictEqual(com1.eq(com2), true)
```

Commitment example

```
  Commitment
cde356c044a12a090c6f48bdc8c90e5b945b8ecc081e3e414061601089f77f05
cde356c044a12a090c6f48bdc8c90e5b945b8ecc081e3e414061601089f77f05
    ✓ It should collide!


  1 passing (9ms)
```

Same problem different places

Those kind of constructions are used a lot in practice:

- Merkle trees
- MPC especially threshold signatures scheme (TSS)
- Zero Knowledge proofs

Practical attack

Last year, Kudelski Security was hired by io.Finnet to audit their modified version of BNB-Chain's tss-lib. Kudelski Security reported to io.Finnet the same hash collision issue again due to concatenating input values with delimiter '$'. The issue this time got mitigated by io.Finnet in a more elegant way and later publicly disclosed as CVE-2022-47931 on Mar 28, 2023.

research.kudelskisecurity.com/2023/03/23/multiple-cves-in-threshold-cryptography-impl...

CVE-2022-47931: Collision of hash values

The functions SHA512_256 and SHA512_256i are used to hash bytes or big integer tuples, respectively. They take as input a list of values and output a hash. According to the paper, those hash functions should behave like a random oracle, and thus it should not be easy to find collisions.

The issue we found arises when hashing multiple concatenated input values, for example, a list of bytes ["a", "b", "c"]. The two vulnerable functions concatenate the values by adding a separator "$" between each value to obtain the string "abc". Then this string is passed to the hash function SHA-512/256 to obtain the hash result. However, the character "$" may itself be part of the input values, so this construction is prone to collisions. As an example, the two input byte array tuples ["a$", "b"] and ["a", "$b"] output the same hash value.

Kudelski Security/io.Finnet's Security Advisory, 2023

TSShock details

In a ECDSA TSS, a multiplicative to additive protocol (MtA) is used:

- The attacker receives: $z = g^x h^y \mod N$
- x and y are unknown and secret
- All other values are controlled by the attacker
- Verifier needs a valid proof that the discrete logarithm between h and g mod N exists.
- If x is found then the private key of the other participant can be recovered.

Proof of knowledge of discrete log

Public input: g and $h = g^x \bmod N$

Private input: a number $x \in \mathbb{Z}_{\phi(N)}$

Prover | Verifier

$\rho \in \mathbb{Z}_{\phi(N)}$

Prover → Verifier: $\alpha = g^\rho \bmod N$

Verifier → Prover: $c \in \{0, 1\}$

Prover → Verifier: $\tau = \rho + cx \bmod N$

Verifier: $g^\tau \stackrel{?}{=} \alpha h^c$

Proof of knowledge

An adversary can cheat the previous protocol with probability ½ thus we need to repeat the protocol 128 times to achieve a security level of 128 bits.

Non interactive proof of knowledge

Public input: g and $h = g^x \bmod N$

Private input: a number $x \in \mathbb{Z}_{\phi(N)}$

Prover

$\rho_i \in \mathbb{Z}_{\phi(N)}$

$\alpha_i = g^{\rho_i} \bmod N$

$c_0, \ldots, c_{127} = \mathrm{H}(g, h, N, \alpha_0, \ldots, \alpha_{127})$

$\tau_i = \rho_i + c_i x \bmod N$

$\alpha_0, \ldots, \alpha_{127}, \tau_0, \ldots \tau_{127}$ →

Verifier

$c_0, \ldots, c_{127} = \mathrm{H}(g, h, N, \alpha_0, \ldots, \alpha_{127})$

$g^{\tau_i} \stackrel{?}{=} \alpha_i h^{c_i} \quad \forall i \in \{0, 127\}$

α-shuffle attack

Since $H(g, h, N, \alpha_0, \ldots, \alpha_{127}) = H(g|h|N|\alpha_0|\ldots|\alpha_{127})$

We can compute $\beta = \mathrm{int}(\alpha|\alpha)$ and $h = \dfrac{\alpha}{\beta}$

Then:

$$H(g, h, N, \ldots, \alpha, \alpha, \ldots) = H(g, h, N, \ldots, \beta, \ldots)$$

α-shuffle attack

Then assign the values of α and β to have a correct proof:

$$c = H(g, h, N, \alpha, \ldots, \beta, \alpha, \ldots, \beta)$$

Then the prover gets:

$$g^{\tau} = \begin{cases} \alpha & \text{if } c_i = 0 \\ \beta h = \beta \frac{\alpha}{\beta} = \alpha & \text{if } c_i = 1 \end{cases}$$

α-shuffle attack

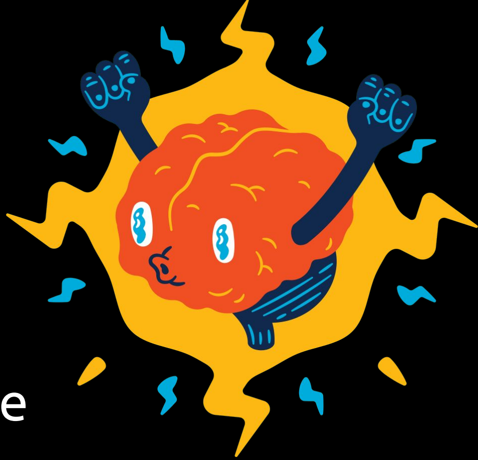
With a forged proof we can send $h = 1$ and finally recover x, by computing the discrete log modulo N.

The private key of the other participant is recovered.

Proof of concept

Train yourself

Training platforms



- **CryptoHacks:** online platform
- **Hackropole**: past challenges of the France Cybersecurity Challenge
- **Donjon CTF** by Ledger (replaced by the SSTIC challenge in 2023)
- **ZK Hack IV:** From 16th January to 6th February 2024. (Past challenge solutions are available)
- **Eurocrypt** 2024 workshop

Eurocrypt 2024 workshop

Workshop on Crypto Code Audit + Capture the Flag:

One day workshop, morning presentations and afternoon dedicated to a small capture the flag competition.

Conclusion

- Crypto code audits are important
- They cost but add lot of value
- They never offer 100% guarantee
- Require a skill mix of both theoretical crypto and implementation
- Human factors can influence the outcome
- Come to learn more at Eurocrypt 2024 in Zurich !

Thank you!

KUDELSKI SECURITY

Links

- GG20 paper: https://eprint.iacr.org/2020/540
- Attacking threshold wallets: https://eprint.iacr.org/2020/1052.pdf
- TSShock: https://www.verichains.io/tsshock/
- Cryptohack: https://cryptohack.org/
- Hackropole: https://hackropole.fr/en/
- Donjon CTF: https://ctftime.org/ctf/547/
- ZK Hack: https://zkhack.dev
- Eurocrypt workshop: https://eurocrypt.iacr.org/2024/affiliated.php