taiko

Crafting your cryptographic library from scratch

Mamy Ratsimbazafy

ZK Engineering @ Taiko

mratsim

m_ratsim

A quick technical presentation of Taiko

- **Type 1 zkEVM (Ethereum-equivalent)** → no changes to hashes, state trees, transaction trees, precompiles or any other in-consensus logic. Full implementation of the Ethereum (execution layer) yellow paper specifications

- **Based rollup** → block sequencing driven by L1 validators, thereby inheriting Ethereum's level of decentralization

- **Permissionless** → proposers & provers can join or leave the network at any time, thereby maximizing censorship resistance

- **Deterministic block execution** → finality is achieved immediately after a block is proposed, i.e. all block properties are immutable from that point on. Thereby Taiko L2 transactions are finalized after only a single L1 confirmation

Who am I?

- **Ethereum Core Dev on Nimbus client**
  - Contributed to IETF standardization of hashing-to-elliptic-curve and BLS signatures
  - Contributed to Ethereum cryptographic test suites
  - Implemented all Ethereum consensus cryptographic protocols

- **ZK Engineering lead at Taiko**
  - Accelerating ZK primitives:
  - Number-theoretic acceleration
  - High-performance computing style acceleration (CPU caches, memory bandwidth, parallelism)
  - Hardware-acceleration

What will we cover today?

The unknown unknowns

- The more you know, the more you realize you don’t know.
  See also Dunning-Kruger effect.
- Signposts for your journey, keywords to search for.
- Dealing with imposter syndrome

Agenda (1/2)

1. My journey into Cryptography
2. The meaning of “Don’t roll your own crypto”
    - Schneier’s Law
    - Design bugs
    - Managing expectations
3. Before your journey
    - What can cryptography do?
    - Threat models
    - Misuse resistance
    - Side channel attacks
    - Picking your programming language

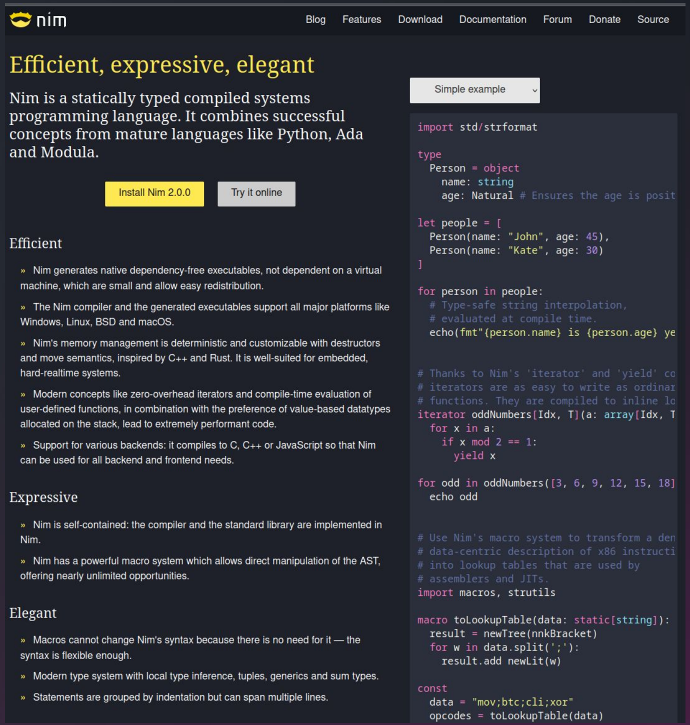
Agenda (2/2)

4. Mapping the journey
   - What to implement?
   - Non-algebraic cryptography
   - Algebraic cryptography
   - Dealing with math-heavy papers or specs
5. Writing your implementation
   - Prototypes
   - SageMath: Constants & Test vectors
   - Debugging cryptography
   - Testing efficiently for edge cases
   - Benchmarking
6. Community
   - The importance of finding a community
   - Showcasing your work
   - Receiving critics

My 5-year journey into cryptography

2018 - Ethereum client in “Nim”

https://nim-lang.org/



2018 - Ethereum client in "Nim"

https://nim-lang.org/

- No production-grade:
  - big integers
  - Networking
  - Cryptography
- But easy C/C++ interoperability
- No supply chain attack

2018 - Ethereum client in “Nim”

https://nim-lang.org/

- Wrapped C++ big integers
- Implemented big integers from scratch

- *Failed* to implement ECDSA for secp256k1
- Wrapped bitcoin/libsecp256k1

2020 - Ethereum client in “Nim”

https://github.com/status-im/nim-blscurve

- Hash-to-Elliptic curve for BLS12-381
- BLS signatures
- Contribution to IETF standardization

- Backends: Milagro/Miracl and BLST
- Security audits
- Multithreading for batch verification

Since 2020 - Constantine

https://github.com/mratsim/constantine, restarted my 2018 failure as a personal project

- BLS signatures
  - Fastest scalar-mul (constant-time)
  - Fastest verification
- EIP 4844 - Protodanksharding
- Top 1 or top 2 with Gnark on performance
  - Fastest single-threaded MSM
- Fuzzing sponsored by the Ethereum Foundation.
  - Now part of Google cryptofuzz
- Verkle trees sponsored by EF Fellowship Program

The meaning of “Don’t roll your own crypto”

Schneier's Law

https://www.schneier.com/blog/archives/2011/04/schneiers_law.html

- "Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break."
- "any person can invent a security system so clever that she or he can't think of how to break it."
- "Few false ideas have more firmly gripped the minds of so many intelligent men than the one that, if they just tried, they could invent a cipher that no one could break."

“Don’t roll your own crypto”

- is not about preventing from from implementing standards, it’s about coming up with novel ways (to be broken)
- is about learning but not trusting yourself in production

“Don’t roll your own crypto”

Seasoned cryptographers are also faillible

Paper 2023/969

Revisiting the Nova Proof System on a Cycle of Curves

Wilson Nguyen, Stanford University
Dan Boneh, Stanford University
Srinath Setty, Microsoft Research

Abstract

Nova is an efficient recursive proof system built from an elegant folding scheme for (relaxed) R1CS statements. The original Nova paper (CRYPTO'22) presented Nova using a single elliptic curve group of order p. However, for improved efficiency, the implementation of Nova alters the scheme to use a 2-cycle of elliptic curves. This altered scheme is only described in the code and has not been proven secure. In this work, we point out a soundness vulnerability in the original implementation of the 2-cycle Nova system. To demonstrate this vulnerability, we construct a

“Don’t roll your own crypto”

Misusing components, even if secure individually, may create MORE vulnerabilities

Paper 2023/329

Caveat Implementor! Key Recovery Attacks on MEGA

Martin R. Albrecht, King's College London
Miro Haller, ETH Zurich
Lenka Mareková, Royal Holloway University of London
Kenneth G. Paterson, ETH Zurich

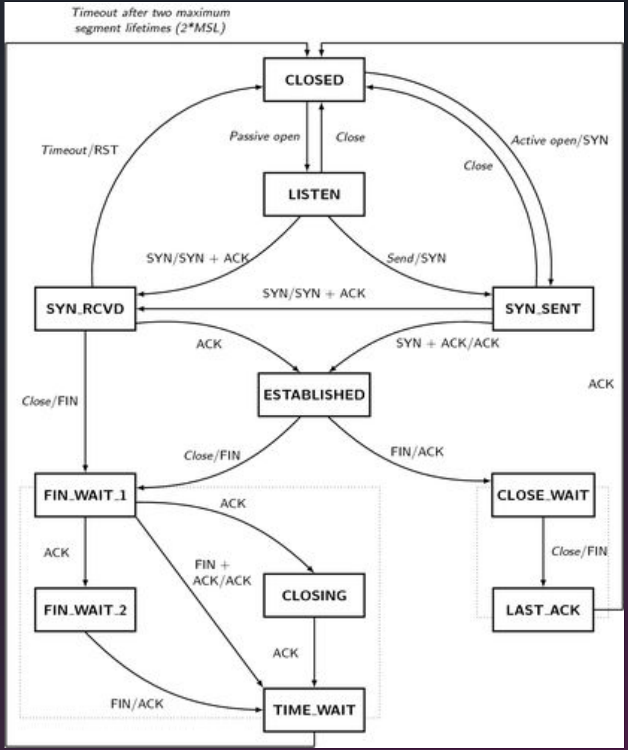
Abstract

MEGA is a large-scale cloud storage and communication platform that aims to provide end-to-end encryption for stored data. A recent analysis by Backendal, Haller and Paterson (IEEE S&P 2023) invalidated these security claims by presenting practical attacks against MEGA that could be mounted by the MEGA service provider. In response, the MEGA developers added lightweight sanity checks on the user RSA private keys used in MEGA, sufficient to prevent the previous attacks.

We analyse these new sanity checks and show how they themselves can be exploited to mount novel attacks on MEGA that recover a target user's RSA private key with only slightly higher attack complexity than the original attacks. We identify the presence of an ECB encryption oracle under a target user's master key in the MEGA system; this oracle provides our adversary with the ability to partially overwrite a target user's RSA private key with chosen data, a powerful capability that we use in our attacks. We then present two distinct types of attack, each type exploiting different error conditions arising in the sanity checks and in subsequent cryptographic processing during MEGA's user authentication procedure. The first type appears to be novel and exploits the manner in which the MEGA code handles modular inversion when recomputing $u = q^{-1} \bmod p$. The second can be viewed as a small subgroup attack (van Oorschot and Wiener, EUROCRYPT 1996, Lim and Lee, CRYPTO 1998). We prototype the attacks and show that they work in practice.

Design bugs

TCP connection “state machine”



Design bugs

Formal verification

- Most programming languages cannot help you on a design bug.
- Borrow checking is not enough, formal verification is necessary, as used in hardware design, medical industry and aerospace industry. (Example Ada/Sparks)

- 2-phase commit protocol for databases
- “Critical sections” without locks
- Therac radiotherapy machine
  - https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/

Design bugs

Managing expectations

Do not misrepresent or omit from your project README that your code is experimental, unaudited and may launch missiles and eat puppies

Before your journey

What can cryptography do?

- ❖ Authentication, ensuring you communicate with the right person
  - ➢ digital signatures
- ❖ Integrity, a received message has not been tampered with
  - ➢ Hash functions
- ❖ Confidentiality, only the intended recipient can read a message
  - ➢ Encryption
- ❖ Non-repudiation, proving that a sender sent a message
- ❖ Plausible deniability, leaving reasonable doubt that an action was not taken

- ❖ Or a combination thereof (authenticated encryption, traitor tracing, ZK …)

Threat models

Cryptography engineering, contrary to many engineering disciplines cannot assume goodwill. It's adversarial.

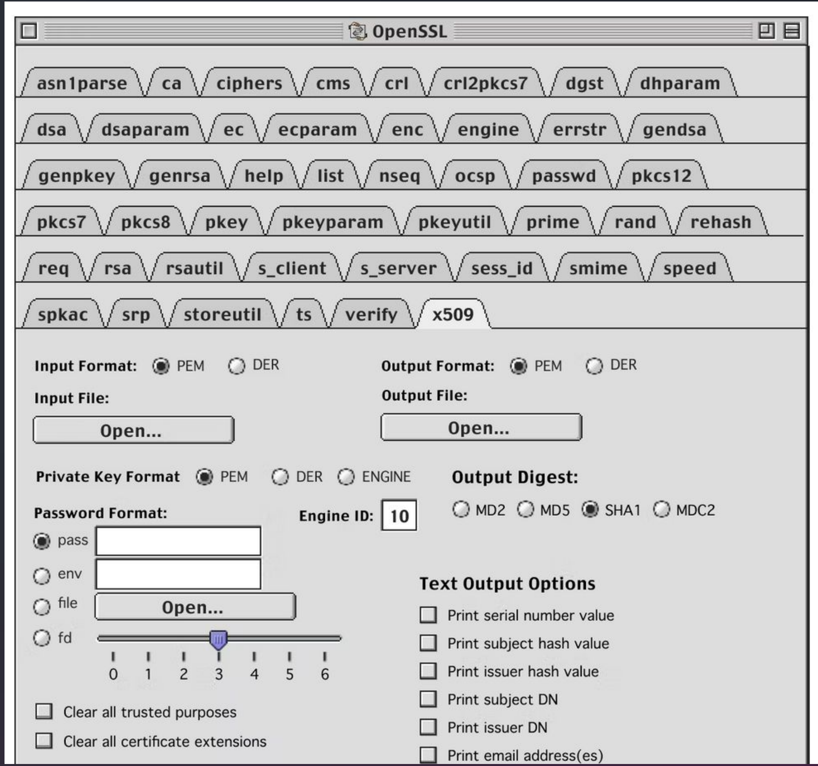
Enter Eve (an eavesdropper) and Mallory (a malicious party).

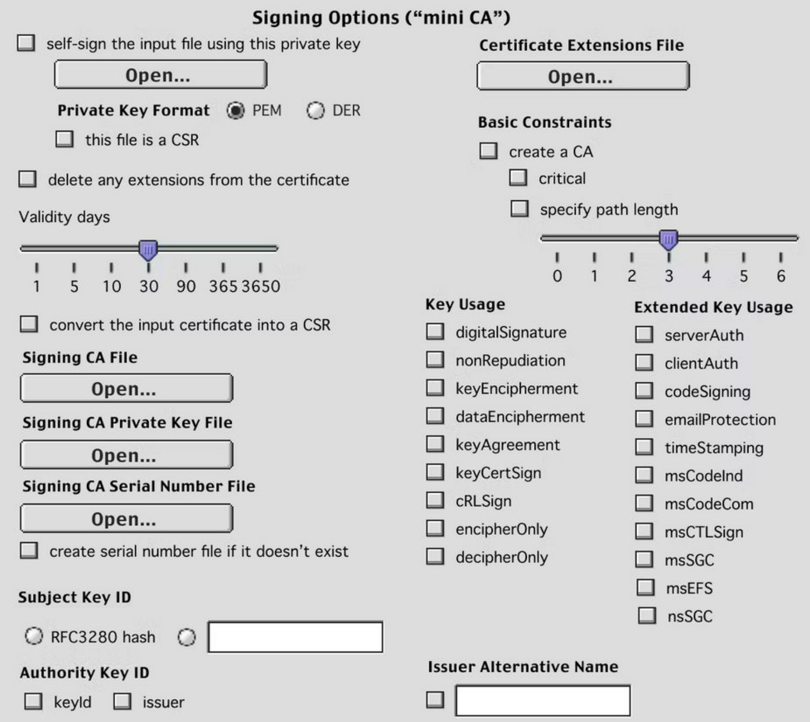
What attacks are you defending against? From who? How much money can they throw at you to break your scheme?

- Information-theoretic security
- computational security
- crypto-economic security

Misuse-resistance

https://smallstep.com/blog/if-openssl-were-a-gui/





Side-channel attacks, constant-time

https://github.com/mratsim/constantine/wiki/Constant-time-arithmetics

- WPA3
- Laptops' TPM (Trusted Platform Module)
- Intel SGX via thermometer
- TLS 1.2 and 1.3
- Libgcrypt and GPG

- Timing attacks (can be done remotely)
- CPU-cache timing attacks
- Power analysis attacks
- Electromagnetic Emission attacks

- Non-crypto (but privacy attack): Twitter Silhouette
  https://blog.twitter.com/engineering/en_us/topics/insights/2018/twitter_silhouette

Picking your language

- ❖ Start with one you’re comfortable with or that is “easy” to learn.
- ❖ Too much crypto and programming language problems (Haskell, Rust, …) is a recipe for burnout

- ❖ Low-level crypto needs precise control over memory and assembly for security. (Compilers can defeat non-assembly seemingly constant-time construct)

Picking your language

- Start with one you're comfortable with or that is "easy" to learn.
- Too much crypto and programming language problems (Haskell, Rust, ...) is a recipe for burnout

- Low-level crypto needs precise control over memory and assembly for security. (Compilers can defeat non-assembly seemingly constant-time construct)

Mapping the journey

What to implement?

Be a user first, pick a program or a library, find what works, what doesn’t.

Non-algebraic cryptography

No math!

- Ciphers
- Hashes
- MAC (Message Authentication Code)
- KDF (Key Derivation Function)
- Random Number Generators

- Follow the spec and official vector
- Differential fuzzing against a reference implementation

Algebraic cryptography

Lots of math!

The base of public-key cryptography / asymmetric cryptography / trapdoor functions

- ❖ Finite Fields
- ❖ Elliptic Curves
- ❖ Pairings
- ❖ Polynomial Commitments
- ❖ Proof systems

- ❖ Lattices

Dealing with math-heavy papers or specs

- ❖ Read the abstract or overview of the goal
- ❖ Model as a black-box first, function over form
- ❖ Math doesn’t bite *(but it might give headaches)*
- ❖ Find an implementation you can test with
- ❖ Prototype

Writing your own implementation

README

- Experimental disclaimer (missile, puppies, ...)
- Security disclosure process (what address, PGP key?)

SageMath

```
def genScalarMulG1(curve_name, curve_config, count, seed, scalarBits = None):
  p = curve_config[curve_name]['field']['modulus']
  r = curve_config[curve_name]['field']['order']
  form = curve_config[curve_name]['curve']['form']
  a = curve_config[curve_name]['curve']['a']
  b = curve_config[curve_name]['curve']['b']

  Fp = GF(p)
  G1 = EllipticCurve(Fp, [0, b])
  cofactor = G1.order() // r

  out = {
    'curve': curve_name,
    'group': 'G1',
    'modulus': serialize_bigint(p),
    'order': serialize_bigint(r),
    'cofactor': serialize_bigint(cofactor),
    'form': form
  }
  if form == 'short_weierstrass':
    out['a'] = serialize_bigint(a)
    out['b'] = serialize_bigint(b)
```

SageMath

```
vectors = []
set_random_seed(seed)
for i in progressbar(range(count)):
  v = {}
  P = G1.random_point()
  scalar = randrange(1 << scalarBits) if scalarBits else randrange(r)

  P *= cofactor # clear cofactor
  Q = scalar * P

  v['id'] = i
  v['P'] = serialize_EC_Fp(P)
  v['scalarBits'] = scalarBits if scalarBits else r.bit_length()
  v['scalar'] = serialize_bigint(scalar)
  v['Q'] = serialize_EC_Fp(Q)
  vectors.append(v)

out['vectors'] = vectors
return out
```

SageMath

- Prototype
- Debugging
- Generate constants
- Test vectors

Testing

- ❖ Test vectors

- ❖ Property-based testing
- ❖ Fuzzing
  - ➢ Poor man's fuzzer: https://github.com/mratsim/constantine/issues/53
- ❖ Differential fuzzing

Benchmarking

- Time
- Cycle
- Throughput (nanoseconds per operations)

- Turbo / Hyperthreading
- Using cloud instances

Community

Community

- ❖ Finding a welcoming community
- ❖ Showcasing your work
- ❖ Receiving critics
- ❖ Receiving critics from cryptographers

The End

taiko

A cryptographer joke:

"Cryptography is the art of transforming a security problem into a key management problem"

taiko

Crafting your cryptographic library from scratch

Mamy Ratsimbazafy

ZK Engineering @ Taiko

mratsim

m_ratsim