ZK is Crypto: 10 Mistakes You Probably Make and How to Avoid Them

Dmitry Khovratovich

Myself

Sikoba - *Researcher*

Dusk - *Senior Cryptographer*

Evernym - *Principal Cryptographer*

ABDK Consulting - *Founder*

Khovratovich@gmail.com

Broke 20+ cryptographic schemes (3 at the time of presentation)

Deanonymized Bitcoin

Designed Argon2 (password hashing standard), Equihash and MTP (Proofs of work in 10+ schemes), Poseidon and Starkad (hashes for zero knowledge).

Consulting for the last 10 years.

Audited 50+ smart contracts, found flaws worth of $10mln

Mindset

Building crypto is not just building a regular application, it is rather a *carcasse* of your building

- Higher risks
- Higher costs
- Difficult to repair
- If there is problem you will know only after everything gets broken

Mistake 1.

Invent your own crypto

- Creating a new crypto is surprisingly hard, error prone and is a big risk

Mistake 1.

Invent your own crypto

- Creating a new crypto is surprisingly hard, error prone and is a big risk
- 80% crypto designed by cryptographers gets broken, this rate reaches 99% for non cryptographers.
- Sometimes you do need your own crypto, for example you need a new protocol. Then hire a cryptographer.

Mistake 2.

Security by obscurity

- Design an algorithm/protocol, but never tell the details?

Mistake 2.

Security by obscurity

- Design an algorithm/protocol, but never tell the details?
- Wrong practice: scheme details are often easy to reverse-engineer.
- The more third-party review you get, the better protocol you obtain.
- After centuries, cryptographers have come to a conclusion: “*the only secret part of a scheme should be the key*”.

Protocol Design

Sometimes you need a new protocol tailor-made for your business.

You checked there is no analogue available.

Mistake 3.

One cryptographer is enough

- You hired a good cryptographer, gave him the task and money, and are now waiting for the result.

Mistake 3.

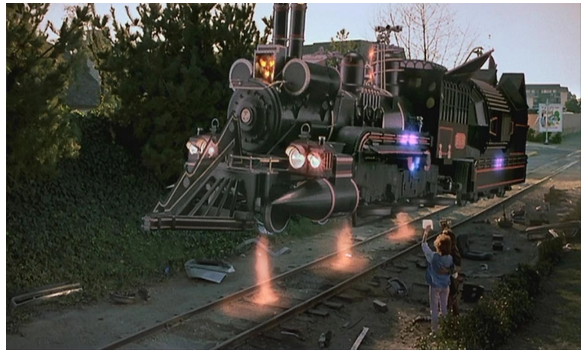
One cryptographer is enough

- You hired a good cryptographer, gave him the task and money, and are now waiting for the result.
- Apparently, even best cryptographers never work alone.
- A big part of security comes from the public scrutiny, so
    1. Review the crypto design with other team members, even not cryptographers.
    2. Use rewards or competitions if you can.
    3. Prefer clarity and brevity to complexity.

Mistake 4.

It works so it is good

1. You mentioned a problem to a friend.
2. In a week, some unknown suddenly comes with a solution.
3. It works and is reasonably fast. Should you accept it?



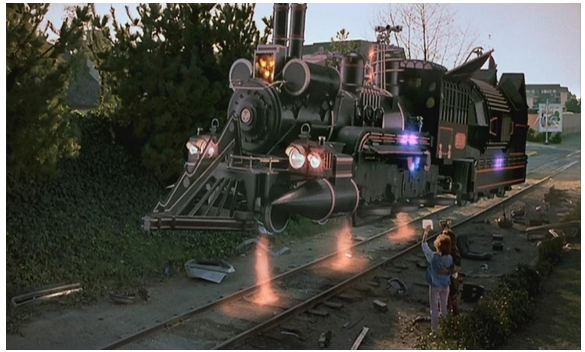
Mistake 4.

It works so it is good

1. You mentioned a problem to a friend.
2. In a week, some unknown suddenly comes with a solution.
3. It works and is reasonably fast. Should you accept it?

No, because:

- The solution might not be optimal;
- The solution might be less secure than potential alternatives;
- A week is too short for a good design.



Mistake 5.

Some privacy is enough

Suppose you design a private money transfer system.

- Transfers are not fully anonymous because it would be too expensive.
- A cheaper system allows narrowing down sender candidates to 10,000. Is it good enough?

Apparently not as there are many ways to amplify the privacy loss. It can become 1-of-10.

There is no some privacy, there is privacy or no privacy.

Mistake 5.

Some privacy is enough

Suppose you design a private money transfer system.

- Transfers are not fully anonymous because it would be too expensive.
- A cheaper system allows narrowing down sender candidates to 10,000. Is it good enough?

Mistake 6.

Complex setup

Your team comes up with a fast and secure solution. However, it needs a lot of time and work for initialization. Should you take it?

Mistake 6.

Complex setup

Your team comes up with a fast and secure solution. However, it needs a lot of time and work for initialization. Should you take it?

Probably not. A sophisticated setup may be difficult to repair.

Example:

1) In March 2018 a member of private cryptocurrency Zcash found how to print fake coins. The problem was in the setup phase which was public but revealed too much.
2) Zcash decided to keep the problem secret for a few months till the scheduled new setup, risking an attacker finding the same problem.

Implementation

The problem has been solved: a beautiful and secure protocol has been found, now it is time to implement it.

Mistake 7.

Key stored everywhere

You deal with secret keys in many components. Should you store them everywhere where they are used, to minimize the risks?

Mistake 7.

Key stored everywhere

You deal with secret keys in many components. Should you store them everywhere where they are used, to minimize the risks?

Apparently not. A better solution is to design a special component that stores all keys, and to guard it properly.

Mistake 8.

Old libraries and old coding style

You see a set of well established libraries for cryptography in a familiar programming language. Should you use them?

Mistake 8.

Old libraries and old coding style

You see a set of well established libraries for cryptography in a familiar programming language. Should you use them?

First check the following:

- Support of the primitives (curves or ciphers) that you need.
- Protection from recent attacks (proper key and nonce length, data format, etc.).
- Side-channel protection, if applicable.
- Standard conformance.
- As usual: support and licensing.

Also, open-source does not imply bug absence.

Mistake 9.

Programmers who design and optimize

Good programmers are keen to crypto design, often thinking it is as easy as regular architecture design.

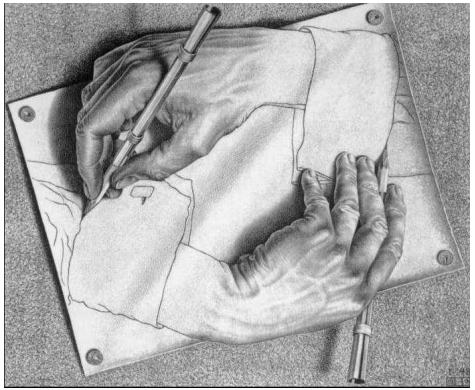
Mistake 9.

Programmers who design and optimize

Good programmers are keen to crypto design, often thinking it is as easy as regular architecture design.

Crypto designed by good coders gets cleaner code but has questionable design decisions.

Cryptographer and programmer should work in pair and review each other’s work.



Mistake 10.

Randomness

Most programming languages have built-in random generators.

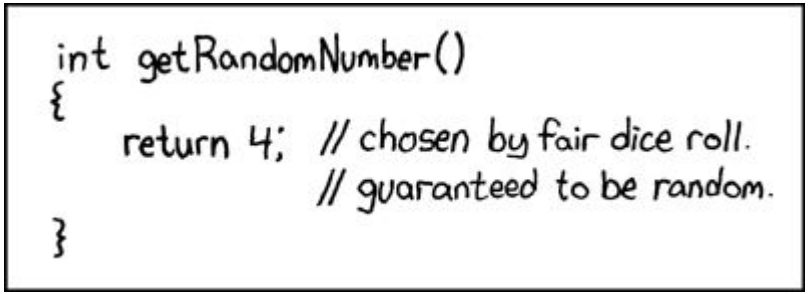
Mistake 10.

Randomness

Most programming languages have built-in random generators.

However, they might be enough for statistical tests only.

- Use one master seed from proper randomness source, ideally a combination of sources.
- Derive other randomness deterministically and one-wayed from the seed.



Summary

1. Do not invent your own crypto
2. Make design public and auditable.
3. One person is never enough.
4. Working does not mean `good`.
5. Little privacy means no privacy. The same for security.
6. Secure your keys in one place.
7. Reevaluate third-party code.
8. Good programmers are not crypto designers.
9. Use cryptographically strong randomness and derive from it.

Questions?