Secure Programming Techniques

Margus Freudenthal, Mart Oruaas

Lecture 5, 2022-03-18
First, Some Formal Matters

- The seminar dates are fixed:
  - 22nd of April
  - 06th of May
  - 20th of May

- Register your seminar topic and date:
  https://docs.google.com/spreadsheets/d/18SBZqa6m5CaGeDXmk1xLaOuaN09Xs8czqHiYm6fmElA/edit?usp=sharing

- Registration deadline is **3rd of April**!
  - Registering after the deadline loses half of the points (5 out of 10)
Cryptography
Setting the Focus

- On practical aspects, affecting software engineering
- For more in-depth coverage, look at:
  - MTAT.07.017
    (https://courses.cs.ut.ee/2022/appcrypto/spring)
  - MTAT.07.002
    (https://courses.cs.ut.ee/2022/crypto1/spring)
  - MTAT.07.003
    (https://courses.cs.ut.ee/2021/crypto-ii/fall)
- Unfortunately, cryptography is provably inaccessible for software developers, in its current state.
  [Hazhirpasand et al., 2021]
What is cryptography

- “Cryptography is the practice and study of techniques for secure communication in the presence of adversarial behavior.”
- “Cryptography is a tool for replacing one problem with another using mathematics.”
- “…and you usually end up with the problem of key management.”
- Cryptography is not a magical solution to anything!
Cryptographic tools

Cryptographic tools are functions that take some private and public inputs and return some private and public outputs (not everything must always be present)

\[(out_{pub}, out_{priv}) \leftarrow f(in_{pub}, in_{priv})\]

That part is kind of “easy”, but complications for software engineers arise from attaching meaning to those computations.

“[…] establishes a legal framework for electronic signatures, electronic seals, electronic time stamps, electronic documents, electronic registered delivery services and certificate services for website authentication.” (eIDAS)
Complicatedness of Meaning

- When Estonian ID-card was first introduced, a lot of guidance was written about configuring mTLS ID-card authentication for web servers.
- This guidance is still available and pops up when searching for one, but it is dead wrong and unsecure, because the world around it has changed.
- One description of the problem can be found in a RIA report from 2021 [Cybernetica, RIA, 2021, 3.1.4.1] (et).
- Original meaning: “mTLS has been established using an ID-card”
- True meaning: “mTLS has been established with some key that has a certificate issued from certain X.509 hierarchy”
Organizational Problems Cannot be Solved by Technology (Alone)

- I can generate a key pair using exactly the same ECC curve as the ID-card uses
- I can generate a technically valid certificate for this key pair on my own, using my own name and ID code
- I can use the private key of that key pair to generate a mathematically and syntactically correct digital signature file which tells that I gift all my earthly properties to someone else
- Yet, this digital signature has zero value, because it is not bound to any mechanism that would bar me from repudiating it and claiming that some unknown person did this, etc
Meaning in Short

- Establish *why* are you doing something
- Establish *which elements* are in play for solving the underlying problem
- An example: lots of people talk about “creating a digital signature using a certificate”, whereas the correct technical procedure is “creating a digital signature using a private key (while maybe attaching the related certificate to the signature)”
Kerckhoff’s principle

- “Security of a system should only depend on the secrecy of the key” (free reformulation)
- Also: “keep secret only the bits that need to be secret for the security of the system and nothing more”
- One of the key guidelines in cryptographic and security engineering
- Is opposed to “security through obscurity”
- The latter is fine in some situations, but when one is employing obscurity, one must do it consciously and plan for its breakage
Dolev-Yao Security Model

- A model given by Dolev and Yao back in 1983 when analyzing public key cryptographic protocols against active attackers [Dolev and Yao, 1983]
- “The attacker carries the message”
- Very theoretical, not too useful in proving security of actual cryptographic protocols, but very good as a mental model
- Practical protocols that involve public clients, like old versions of OpenID and OAuth have been completely broken because they have not taken the model fully into account
- It has significant influence on the “do not roll your own cryptography” stance
Security Level

- “n-bit key” as a security measure is a thing of the past, leave it to bad science fiction writers
- Security of modern cryptographic algorithms is these days measured in “bits”: $b$-bit security of an algorithm means that it takes approximately such a computing resource to break the algorithm that is required for performing $2^b$ block cipher operations [NIST, 2020, 5.6.1.1]
- Security levels to be used are usually prescribed: by regulations or by local company policies
- Security level estimates are published by authoritative bodies [NIST, 2020] [Cybernetica, RIA, 2021]
Rubber Hose Cryptoanalysis

- Instead of breaking the raw cryptography, it is usually easier to go around it
- As usual, humans are the weakest link in the chain: threats, carelessness, bribery
- “Beat him with this piece of rubber hose until he reveals the password” (5-dollar wrench works as well: XKCD #538)
- This is why procedures, physical security, software quality and human well-being matter
- Cryptography is always part of a system.
Do Not Roll Your Own Cryptography

- Schneier’s Law: “Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can’t break. It’s not even hard. What is hard is creating an algorithm that no one else can break, even after years of analysis. And the only way to prove that is to subject the algorithm to years of analysis by the best cryptographers around.”

- Also, implementations can go horribly wrong: https://fermatattack.secvuln.info/

- The title of this slide points to both cryptosystems’ design and their implementations
What Can Go Wrong, Even Worse

- A seemingly simple “extra security” addition to the protocol changes the protocol’s nature completely, allowing an active adversary to compromise the whole traffic flow.
- It is pretty common that mis-designed or mis-implemented cryptosystems are breakable by middle-school math: complex systems tend to fail catastrophically.
Do Roll your Own Cryptography, But...

- Someone started it all, so they rolled their own cryptography
- Good things come out of such tinkering, like WireGuard
- Be extremely careful, start with throwaway experiments
- Seek reviews and cooperation
- Spend years on this before putting your first piece to production
- A good read: https://loup-vaillant.fr/articles/rolling-your-own-crypto
Interoperability vs internal use

- There only a handful of really interoperable cryptographic protocols that a software developer usually meet in their work: TLS, OAuth, OpenID Connect most notably.
- Any application that cannot be satisfied by such public tools has to turn to in-house solutions, unfortunately.
- There is pretty much no standard for elementary things like encrypting small amounts of data for temporary storage.
- In order to solve those problems, one should turn to high quality opinionated libraries like Libsodium or Tink and hire an expert.
Where to learn from?

- Finding good cryptography-oriented materials for software developers is not easy
- As mentioned, the whole scene is even hostile towards mainstream software developers [Hazhirpasand et al., 2021]
- Cryptopals challenge: https://cryptopals.com/
- There is a fresh JavaScript-oriented book by Alessandro Segala [Segala, 2022]
- Documentation of modern libraries: TweetNaCl, Libsodium, Tink.
Why Centralized Control is Almost Always Inevitable

- The need for attaching meaning to cryptographic elements requires at least some amount of centralized control which allows to coordinate that meaning.
- In real life: X.509 CA PKI, directory services like LDAP and Active Directory, not to mention all kinds of organization policies and procedures.
- Trying to avoid systematic control is very expensive.
- Chaotic management of cryptographic materials will very likely lead to vulnerabilities.
Key Management

- Key Management is usually the topic a software engineer meets most often when dealing with cryptographic solutions on the field.
- Alas, it is also one of the most underappreciated and underdocumented fields.
- Rule of thumb #1: separate key (secret) handling from the rest of application logic. One very common way of compromising the secrets is publishing them in a source code repository.
- Rule of thumb #2: build it so that all secrets have short lifetimes. See Let’s Encrypt example.
Recommendations

- Work with an expert. At least, work together with SOMEONE. Peer review matters
- Use well-vetted libraries
- Try to separate cryptographic problems from implementation details, think in terms of cryptographic messages and operations
- Separate code modules dealing with cryptography from the rest
- Get the key management right!
Questions?
References I


References II


References III
