Introduction to cryptology (GBIN8U16)

✧

Introduction

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html

2021–02–03

First things first

Main goals of this course:

- Motivate the field (why is cryptography useful?)
- Introduce some constructions (what's a block cipher, a key exchange?...)
- Introduce some attacks (how do you find collisions for a random function?...)
- Introduce some real-life usage (e.g. TLS)

Schedule

Previous slide in order:

- Definitions and basic security notions for:
  - Block ciphers, symmetric encryption, MACs, hash functions
  - Discrete log-based key exchange & signatures, RSA (incl. paddings)
- A few examples of generic attacks
- A few concrete use-cases/applications/attacks

Organisation

There will be:

- Lectures (such as this one)
- Tutorial sessions (mostly)
- Practical/lab sessions (occasionally)
- A contrôle continu evaluation (a small programming project)
- A final exam

What's crypto?

Quick answer: it's about protecting secret data from *adversaries*

- In a communication (encrypted email, text messages; on the web; when paying by credit card)
- On a device (encrypted hard-drive)
- During a computation (online voting)
- Etc.

Where does crypto run?

Crypto needs on various platforms

- High-end CPUs (Server/Desktop/Laptop computers,...)
- Mobile processors (Phones,...)
- Microcontrollers (Smartcards,...)
- Dedicated hardware (accelerating coprocessors, cheap chips,...)

Techno constraints

Varying contexts, varying requirements

- Speed (throughput)
- Speed (latency)
- Code/circuit size
- Energy/power consumption
- Protection v. physical attacks

$\Rightarrow$ Implementation plays a big part in crypto

Quick example

A protocol (e.g. TLS) uses among others

- A key exchange algorithm (e.g. Diffie-Hellman) — "public-key" cryptography
  - instantiated with a secure group (e.g. ANSSI FRP256V1)
- An authenticated-encryption mode of operation (e.g. GCM) — "symmetric-key" cryptography
  - instantiated with a secure block cipher (e.g. the AES)
- A digital signature algorithm (e.g. ECDSA) — "public-key" + "symmetric-key" cryptography
  - instantiated with a secure group and a secure hash function (e.g. SHA-3)

Protocols can be complex

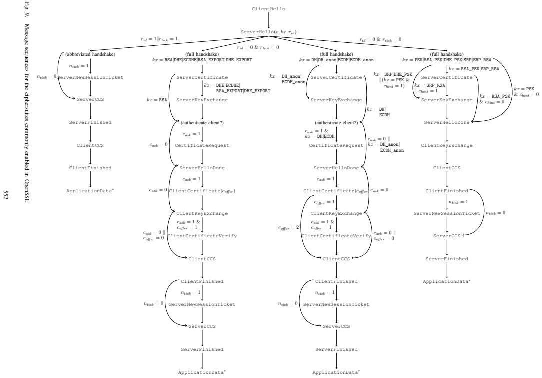

Figure: Part of the TLS state machine, Beurdouche et al., 2015

“Doing crypto”

- Designing new primitives/constructions(/protocols)
- Analysing existing primitives/...
- Deploying crypto in products

- Different goals, different techniques
  - Ad-hoc analysis, discrete mathematics, algorithmics
  - Computational number theory/algebraic geometry
  - Low-level implementation (assembly, hardware)
  - Formal methods
  - Following “good practice”

Scope of an analysis

Many types of adversary

- Passive ("eavesdropper = Eve")
- Not passive, i.e. active
- With or w/o physical access
  - Side channels
  - Fault attacks
- With varying scenarios ("one-time" or long-term secret?)
- With varying objectives

Security objectives?

Security objectives?

- Hard to find the “keys”
- Hard to find the message (confidentiality)
- Hard to change/forge a message (integrity/authenticity)
- Etc.

Remark

Most of the time, one aims for some form of *computational security*: it is always possible to break everything by spending “enough” time $\rightsquigarrow$ just make sure that “enough” is “too much”.

A broader perspective

In crypto (as in science in general), we need:

Figure: Nebular's wisdom (Watterson)

Definitions for science!

It is essential to properly define:

- The objects we use, e.g. what kind of basic *functionality* ("API") is required (so that there's no ambiguity about what we're talking about)
- The properties we want the objects to further satisfy, e.g. what kind of *security* we expect (so that there's no ambiguity about whether we've succeeded or not)

One of the main goals of this course: learn about cryptographic objects AND some associated security properties!

Models are hard

- In crypto, it is common to have *several security models* for a *single* object
- For instance a block cipher may be analysed w.r.t. PRP, SPRP, XRKA-PRP, KCA... security or may further be assumed to be ideal!
- One needs to use a model appropriate for its actual use (symmetric encryption, building a tweakable block cipher, a compression function...)

A quick model example

Indistiguishability in a chosen-plaintext setting (IND-CPA); fair model to decide if $\mathbb{O}$ implements a good symmetric encryption scheme:

1. Submit messages to an *oracle* $\mathbb{O}$ to be encrypted, & get the result
2. Choose, m_0, m_1 of equal length; send both to $\mathbb{O}$
3. Receive $\mathbb{O}(m_b)$ for a random $b \in \{0, 1\}$
4. Goal: determine the value of b (better than by guessing)

- $\mathbb{O}$ has to be *randomized*

A code that's not IND-CPA

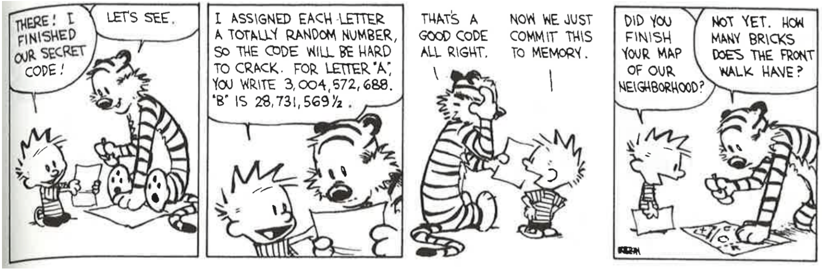

Figure: Calvin & Hobbes' code (Watterson)

Randomness is key in crypto

Random numbers always needed

- To generate keys
- To generate *initialization vectors* (IVs) or *nonces*
- To generate random masks (to protect against some attacks)
- Etc.

Random number generation is not easy

Lead to severe vulnerabilities, several times. For instance:

- Debian's OpenSSL key generation (2006–2008)
- WWW RSA private keys with shared factors (Lenstra et al., 2012)
- Smartcard RSA w/ biased private keys (Bernstein et al., 2013)
- Smartcard RSA w/ biased private keys (Nemec et al., 2017)

How not to generate random numbers

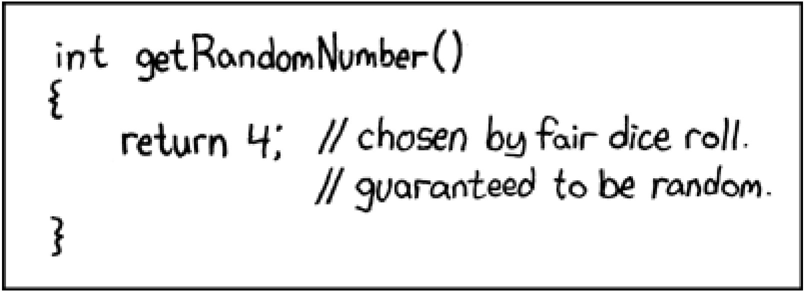

Figure: XKCD's PRNG (Munroe)

How not to generate random numbers

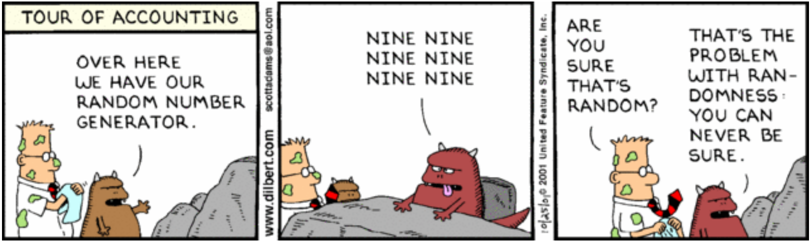

Figure: Dilbert's PRNG (Adams)

Very small Kolmogorov complexity!

How to generate them, then?

A basic idea (e.g. `/dev/urandom`)

- Set up a “random” state (from e.g. physical sources)
- Refresh it continuously as randomness comes by
- Extract and filter when outputs are needed

Are random numbers all you need?

A "perfect" encryption scheme, the one-time pad

1. Let the message m be in $\{0,1\}^n$, n maybe large (say, 2^{40})
2. Let the key k be $\twoheadleftarrow \{0,1\}^n$
3. The ciphertext $c = m \oplus k$

- Knowing c does not give information about m (see TD)

Problems:

- Integrity not guaranteed. So actually NOT perfect in presence of *active* adversaries (i.e. all the time)
- Needs very large keys
- Needs "perfect" randomness too!

What do you need then? Symmetric primitives!

- Stream ciphers (computational variants of OTP), e.g. RC4 (broken), Trivium...
- Block ciphers (encrypt "blocks"), e.g. AES
- Message authentication codes (MACs, check authenticity), e.g. {A,B,C,D,E,F,G,H,I,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Z}MAC (For more on the topic, cf. `https://www-ljk.imag.fr/membres/Pierre.Karpman/JMAC.pdf`)
- Hash functions (securely compress long messages to short digests), e.g. SHA-3

Also need, say, *mode of operations* (to get e.g. IND-CPA)

Complementary primitives: public-key cryptography

Not all primitives need a *single secret key*. One can also have

- Trapdoor permutations (easy to encrypt, hard to decrypt w/o the trapdoor), e.g. RSA
- Public key exchange, e.g. Diffie-Hellman
- Signatures, e.g. DSA

We also need assumptions!

Public-key schemes usually depend on "cryptographic assumptions" (= hardness of some problems), e.g:

- Factorization of large numbers ($\neg$PQ)
- Computing discrete logarithms in $\mathbb{F}_q^\times$, $E(\mathbb{F}_q)$, ... ($\neg$PQ)
- Decoding a noisy codeword of a random error-correcting code (PQ)
- Finding a short vector in a lattice (PQ)
- Solving a quadratic system of equations (PQ)
- "Inverting" hash functions (PQ)
- Etc.

Note: Assumptions can be attacked!

We need keys: secret, private, public...

What are crypto keys like?

- Stream/Block cipher: a binary string
- Hash functions: $\emptyset$
- RSA: a prime number (secret), an integer (public)
- Diffie-Hellman: an integer (secret), a group element (public)
- Code-based: a (generating) matrix (of a linear code) (one secret, one public)
- Etc.

Secrets large and small

What should the secret/public key size be (in bits)?

- Block ciphers?
- RSA?
- Diffie-Hellman (well-chosen $\mathbb{F}_q^\times$)?
- Diffie-Hellman (well-chosen $E(\mathbb{F}_q)$)?
- Code-based (McEliece, Binary Goppa codes)?

Secrets large and small

What should the secret/public key size be (in bits)?

- Block ciphers: e.g. 128 bits
- RSA: e.g. 3072 bits
- Diffie-Hellman (well-chosen $\mathbb{F}_q^\times$): e.g. 3072 bits
- Diffie-Hellman (well-chosen $E(\mathbb{F}_q)$): e.g. 256 bits
- Code-based (McEliece, Binary Goppa codes)? e.g. 200 000 *bytes*

Secrets large and small

What should the secret/public key size be (in bits)?

$\Rightarrow$ Quite a complex matter! (Follow recommendations, e.g. from ANSSI!)

What's 128 bits anyway?

Objective: run a function 2^{128} times within 34 years ($\approx 2^{30}$ seconds), assuming:

- Hardware at 2^{50} iterations/s (that's pretty good)
- Trivially parallelizable (that's not always the case in practice)
- 1000 W per device, no overhead e.g. for cooling (that's pretty good)

$\Rightarrow$

- $2^{128-50-30} \approx 2^{48}$ machines needed
- $\approx 280\,000\,000$ GW 'round the clock
  - $\approx 170\,000\,000$ EPR nuclear reactors

(Of course technology may improve, but this gives quite a safe margin. One must however be careful about the exact attack setting (more of that another day))

That's all for today

Next week:

- Block ciphers: what, why, how?