Information Security

{aeggler,dsparber,jkleine,thgoebel}@ethz.ch

ETH Zürich, FS 2020

This documents is a mélange of a script and a simple list of definitions for the course *Information Security* at ETH Zurich. It contains lots of definitions, a few explanations here and there, and – where applicable – even diagrams! You won't, however, find any involved proofs or lots of examples.

This summary is created during the spring semester 2020. But due to the few changes in syllabus content in the past we have reason to believe that it is also relevant beyond that very semester.

We do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. Feel free to point out any erratas.

Contents

Part I.
Fundamentals of Cryptography

1. Introduction and Definitions

Cryptography was: the art of encrypting messages, now: the science of securing digital communication and transactions

Cryptology = cryptography + cryptanalysis
(= Constructing secure systems + breaking them)

Encryption scheme/cipher encryption and decryption

Provable security needed since there cannot exist an experimental proof that a scheme is secure (no "typical adversary")

Kerckhoff's principle The cipher should remain secure even if the adversary knows the specification of the cipher. The **only** thing that is secret is a (short) key k (usually chosen uniformly at random). If this is not respected we have *security by obscurity*.

Encryption scheme (formally) = pair (Enc, Dec, (Gen?)), where

$$Enc : \mathcal{K} \times \mathcal{M} \mapsto \mathcal{C}$$
$$Dec : \mathcal{K} \times \mathcal{C} \mapsto \mathcal{M}$$
$$\mathcal{K} = \text{Key space}$$
$$\mathcal{M} = \text{plaintext space}$$
$$\mathcal{C} = \text{ciphertext space}$$

i.e. $Enc_k(m)$ and $Dec_k(c)$ are an encryption algorithm and a decryption algorithm respectively[1] and Gen is a key-generation algorithm.

Correctness of an encryption theme

$$\forall k : \ Dec_k(Enc_k(m)) = m$$

1.1. Historical ciphers

Caesar Shift cipher Shift alphabet by key k. Easy to break: check all possible keys (*brute force attack*).
↯ too small key space

[1] For brevity we write $Enc_k(m)$ instead of $Enc(k, m)$

Substitution cipher Any permutation π of the alphabet. Break by using statistical patterns of the language (*frequency analysis*).
↯ same letter == same ciphertext

Vigenere Cipher Poly-alphabetic substitution. Key $k = k_1, k_2, ..., k_d$. Do i-th letter with Caesar using key k_i.
Cryptanalysis: (if d known) identical l-grams are encrypted to same cipher if their distance is $n \cdot d$

1.2. Information-theoretic security

We try to formally define the *security of an encryption scheme*. We write "perfectly secret" but this is equivalent to "perfectly secure".

Perfectly secret (aka. information-theoretically secret or unconditionally secret)

Informally: The adversary should not learn any *additional* information about message m by looking at the ciphertext.

An encryption scheme is **perfectly secret** if for random variables M, C and every $m \in \mathcal{M}$ and $c \in \mathcal{C}$ it holds that[2]:

$$P(M = m) = P(M = m|C = c)$$

$$\Leftrightarrow \text{ M and C are independently distributed}$$

$$\Leftrightarrow^{3} \forall m_0, m_1 : Enc_k(m_0) \text{ and } Enc_k(m_1) \text{ have the same distribution}$$

One-Time Pad Example of a perfectly secret scheme. Encryption / decryption through component-wise xor:

$$Enc_k(m) = k \text{ xor } m = k \oplus m$$

Not practical since 1) the *key* has to be of the same length as the message, 2) it cannot be reused[4] and 3) getting truly random strings is difficult.

Shannon Theorem In every perfectly secret encryption scheme we have

$$|\mathcal{K}| \geq |\mathcal{M}|$$

⇒ One time-pad is optimal in the class of perfectly secret schemes.
⇒ The drawbacks of the one-time pad are unavoidable. In order to achieve something practical, we now limit the power of the adversary.

[2]Note the difference between the plaintext space $\mathcal{M}$ and the random variable M!
[3]Because $P(C = c) = P(C = c|M = m)$
[4]If it were reused, the adversary would learn $m_0 \oplus m_1$ (i.e. the diff of the two messages) which violates perfect security.

1.3. Computational security

Quantum cryptography Security which is based on the laws of quantum physics. Currently not commercially practically.

Indeterminacy: quantum states cannot be measured without disturbing the original state (detectable!).

Computationally-secure cryptography bounds Eve's computational power. Its schemes can in principle be broken if the adversary has huge computing power (or a lot of luck).

Efficiently computable Polynomial-time computable on a Probabilistic Turing Machine[5], i.e. running in time $\mathcal{O}(n^c)$.

System X is (t, ε) secure if every Turing Machine that operates in time at most t can break it with probability at most ε.

Some notation:
M = prob. poly-time TM
M(X) = random variable denoting output of M if tape content was chosen u.a.r.
$Y \leftarrow M(X)$ – Y takes the val that M outputs on input X
$Y \leftarrow \mathcal{A}$ – means that Y is chosen u.a.r. from the set $\mathcal{A}$.

negligible A function $\mu : \mathbb{N} \mapsto \mathbb{R}$ is negligible, if for every natural number c there exists a natural number n_0, such that

$$\forall x > n_0 : \ |\mu(x)| < \frac{1}{x^c}$$

i.e. μ approaches 0 faster than the inverse of any polynomial.

Security parameter n = length of key k. k is a random element of $\{0,1\}^n$. Taken by a scheme X, i.e. we have infinitely many schemes: X(1), X(2), ...

Finding a better definition
Scheme X is secure if

$$\forall \text{ M: P(M breaks the scheme X) is negligible}$$

Pro: All types of TM are equivalent up to a polynomial reduction ⇒ no need to specify details of the model.
Con: asymptotic results tell us nothing about the concrete systems.

[5]Turing Machine with an additional tape of random bits. Note the difference between a Probabilistic and a Non-Deterministic Turing Machine!

Slight *improvement*:
An encryption scheme is perfectly secret if:

- $\forall m_0, m_1, \in \mathcal{M} : Distribution_{Enc(K,m_0)} = Distribution_{Enc(K,m_1)}$ (as above)
- m_0, m_1 are chosen by a poly-time adversary
- no poly-time adversary can distinguish any two distributions with non-negligible probability

Indistinguishably/Semantically/Computationally secure
(Enc, Dec) is an indistinguishable encryption, if any polynomial time adversary guesses b correctly with probability at most $0.5 + \varepsilon(n)$, where ε is negligible and b is whether the oracle returned $Enc(K, m_0)$ or $Enc(K, m_1)$ when given m_0 and m_1.

Reality We need to encrypt multiple messages with the same key. The adversary may learn something by looking at the ciphertexts of some messages m_i
$\Rightarrow$ **Chosen Plaintext Attack CPA**

IND-CPA secure (Enc, Dec) has "indistinguishable encryption under CPA" i.e. is "CPA secure" if every randomized polynomial time adversary guesses b correctly with probability at most $0.5 + \varepsilon(n)$, where ε is negligible.
$\rightarrow$ needs randomisation or keep state
$\rightarrow$ can be implemented in practice

2. Block Ciphers

Theorem IND-CPA encryption w/o hardness assumption exists (with $|k| < |m|) \Rightarrow P \neq NP$

2.1. Pseudo-random functions

Notation

$\mathcal{M} = \{0,1\}^m$ – message space
$F : \{0,1\}^m \mapsto \{0,1\}^m$ – random permutation
$F_k : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ – keyed permutation. F_k and F_k^{-1} poly-time computable.

Problem: Cannot represent random permutation efficiently (exponential space usage) $\Rightarrow$ instead use a function that "behaves almost like random"

Pseudorandom Permutation (PRP) Efficient, deterministic function that returns a pseudo-random output sequence. There exists a efficient inverse function. Cannot be distinguished from a random permutation (by a poly-time distinguisher with non-negligible advantage) – hence the term "pseudo-random".

Pseudorandom Function (PRF) PRP but F_k need not be a permutation.
Notabene: Same security properties. In fact PRP and PRF are indistinguishable for poly-time adversary.

Block Cipher $\approx$ PRP. Specifically: apply PRP to plaintext in blockwise fashion. e.g. DES, AES.

2.2. Block ciphers in practice

Shannon's Confusion and Diffusion Principle

Diffusion: ciphertext bits should depend on plaintext bits in a complex way. If a bit in the plaintext changes any bit of the ciphertext should change with $p = 0.5$.

Confusion: Each bit of the ciphertext should depend on all bits of the key. If one bit of the key changes then the ciphertext should change entirely.

Expansion, Substitution, Permutation

Expansion copies bits to certain output positions to increase output size. Scheme of expansion is known.

Substitution replaces blocks of bits with other (of potentially different size). Replacement pattern is taken from known substitution box (S-boxes). (confuse)

Permutation permutes the bits in a know pattern. (diffuse)
$\Rightarrow$ gives us a function f_i (dependent on k_i) for the Feistel network

Feistel network see Figure 2. To decrypt, reverse the key schedule.

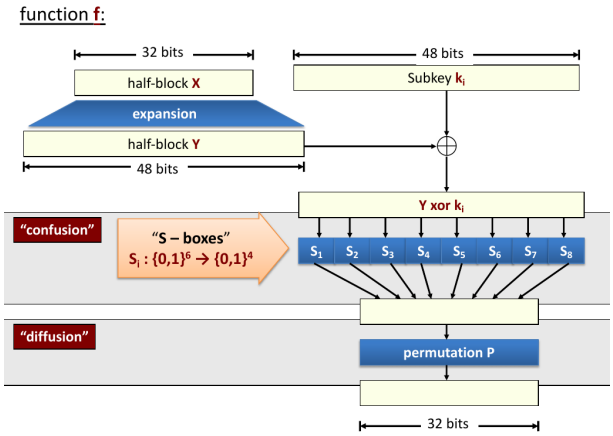

Figure 1: Expansion, substitution and permutation

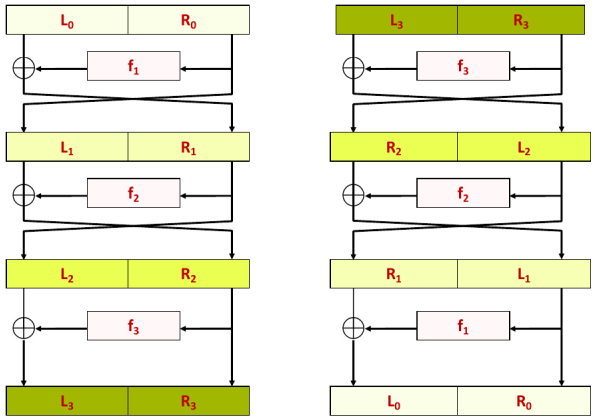

Figure 2: Feistel network. LHS encryption, RHS decryption.

Increasing key size Double encryption ($F'_{(k,k')}(x) = F_{k'}(F_k(x))$) can easily be attacked with *Meet-in-the-middle* attack (given PT/CT pairs, try all possible keys to encrypt half way and all keys to decrypt from the other side, find matching pairs in the list to find valid key).

Instead triple encryption ($F'_{(k_1,k_2,k_3)}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$) is preferred. It is NOT immune to the meet-in-the-middle attack but it takes significantly longer.

Product cipher Cascading (repeatedly applying) a cipher, e.g. to increase key size.

2.3. Block cipher modes of operation

Electronic Codebook (ECB) Every block is encrypted identically $\implies$ same plaintext results in same ciphertext, thus **NOT SECURE**.

Cipher-Block Chaining (CBC) Use initialization vector (IV) to XOR the first plaintext block before applying the actual encryption function, the resulting cipher is used to XOR the plaintext of the next block, and so on. Need to use different IV for every message. The IV and the propagation through the blocks makes sure that the same data does not result in the same cipher.

Theorem: If F is a PRP then F-CBC is secure.[6]

The follow two modes aim at converting a pseudorandom permutation (PRP) into a pseudorandom generator (PRG) G:

Output Feedback (OFB) Use seed and PRF to generate one-time-pad (OTP) to use for first block, use the OTP of the first block to generate the OTP for the second block, and so on. The OTPs are then XORed with the respective message block. Generally not parallelisable, but the OTPs can be precomputed. IND-CPA secure if F is a PRP.

$G(k, IV) = F_k(IV)||F_k(F_k(IV))||F_k(F_k(F_k(IV)))||...$[7]

Counter (CTR) Use $IV + i$ and PRF to generate OTP for respective message block (i.e. OTP for block i is given by $F_k(IV + i)$). IND-CPA secure.

$G(k, IV) = F_k(IV + 1)||F_k(IV + 2)||F_k(IV + 3)||...$

	ECB	CBC	OFB	CTR
Error propagation: transmission error in block c_i affects...	only block c_i	only c_i and c_{i+1}	only block c_i	only block c_i
Encryption parallelisable	✓	×	×	✓
Decryption parallelisable	✓	✓	×	✓
If 1 bit in plaintext changes, recompute ...	1 block[8]	everything	1 block	1 block

Table 1: Comparison of block cipher modes of operation

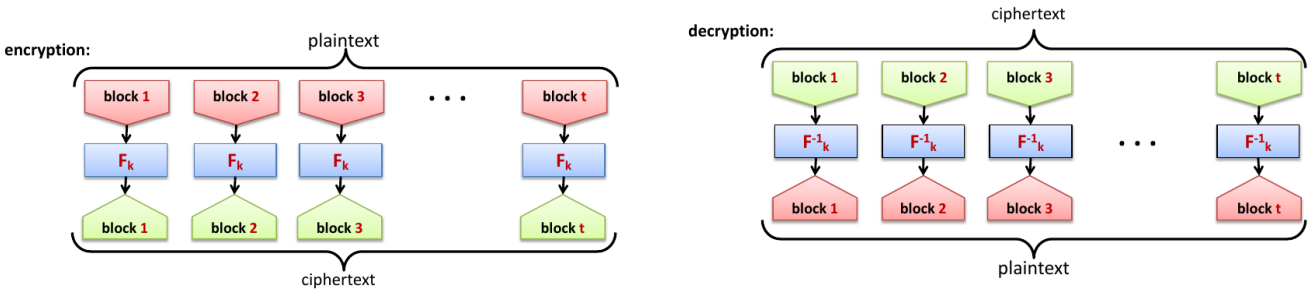

Figure 3: Electronic Codebook ECB

Self-recovering A mode where errors don't propagate, e.g. CBC

Padding If the length of the message is not evenly divisible by the block size, we need to expand the message to fit the block size. This is achieved through adding some data to the end of the message or through *ciphertext stealing*.

Initialisation vector (IV) Random input to a block cipher. Sent to the recipient together with the ciphertext. Enables key reuse.

[6] thgoebel: The slides do not mention which secureness? I assume IND-CPA secure?
[7] The notation || means "concatenation".
[8] Leaks information!

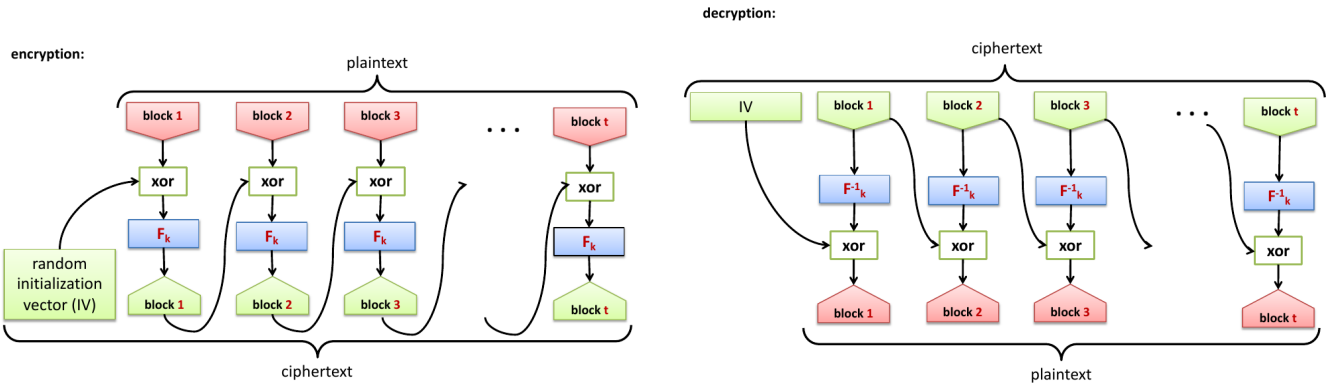

Figure 4: Cipher-Block Chaining CBC

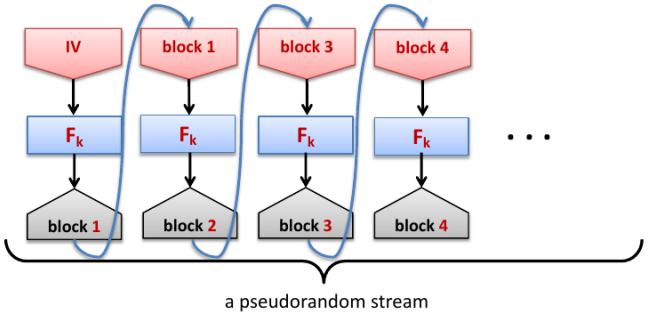

Figure 5: Output Feedback OFB

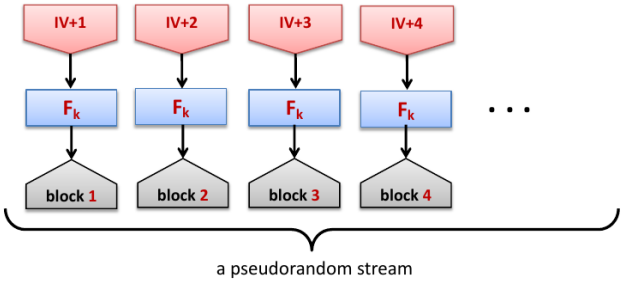

Figure 6: Counter CTR

Seed Input to a PRG. Sometimes this term is used instead of "IV", not always a clear distinction.

Summary graph:
PRP/PRF exists $\xrightarrow{\text{modes of operation}}$ secure encryption exists $\Rightarrow$ one-way functions exist $\Rightarrow$ wraparound: PRP/PRF exists[9]

[9]TODO: Can somebody draw a nice diagram?

3. Stream Ciphers

3.1. Pseudorandom Generators

Pseudorandom Generator PRG $G : \{0,1\}^* \mapsto \{0,1\}^*$ such that

$$\forall n \forall s \text{ with } |s| = n : \ |G(s)| = l(n)$$

where l is a polynomial with $\forall n : \ l(n) > n$ and for a random s then $G(s)$ "looks random" too. s is called the **seed** and l the **expansion factor**.

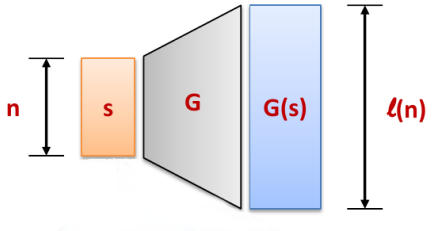

Figure 7: Pseudorandom Generator PRG

Notice that if s is chosen uniform at random (from $\{0,1\}^n$) then $G(s) \in \{0,1\}^{l(n)}$ cannot be u.a.r.

Cryptographic PRG PRG such that every PPTTM can only distinguish $G(S)$ from a R with $p = 0.5 + \varepsilon$; where S is distributed u.a.r. over $\{0,1\}^n$ and R is distributed u.a.r. over $\{0,1\}^{l(n)}$ and ε is negligible.

Golomb's Postulate

1. |Number of 1s – number of 0s in $G(s)$| ≤ 1
2. "Run" of 0s (or 1s) of length i occurs with $p = 2^{-i}$
3. some maths ...

Theorem Cryptographic PRG exists $\implies$ computationally-secure encryption exists (proof by a tight reduction).

One-way function A function $f : \ \{0,1\}^* \mapsto \{0,1\}^*$ is one-way if it is 1) poly-time computable and 2) invertible by a poly-time adversary with negligible probability.

Theorem $P = NP \implies$ one-way functions do NOT exist.[10]
However there are some candidates, like prime factorisation: $f(p,q) = p \cdot q$.

Remark One-way functions do NOT hide all the input! The following is a valid one-way function:

$$f(x_0, ..., x_{n-1}, x_n) = f'(x_0, ..., x_{n-1})||x_n$$

[10]The proof is left as an exercise.

Theorem One of the most fundamental results in symmetric cryptography:

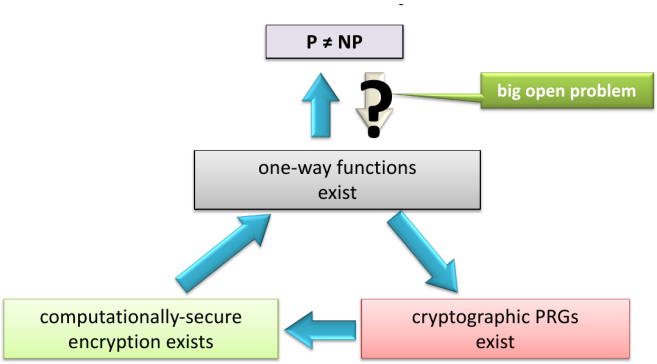

Figure 8: Relation between one-way functions, cryptographic PRGs and computationally-secure encryption

3.2. Stream ciphers in practice

Stream ciphers output "infinite" streams of bits (as opposed to a number of blocks).

The challenge in using PRGs for encryption is that we cannot reuse the same seed - see the one-time pad! So instead:

Synchronised mode Divide $G(s)$ into blocks and xor them with blocks of plaintexts. CPA-secure. Difficult in practise because it requires keeping state (where in $G(s)$ were we?).

Unsynchronised mode Randomise by using a PRG $G(IV_i, s)$ that takes an additional input IV_i (fresh for each message m_i).

For this we can either design such a G from scratch or use a cryptographic hash function H (see chapter 4.2): $G(IV, s) := G'(H(IV||s))$.

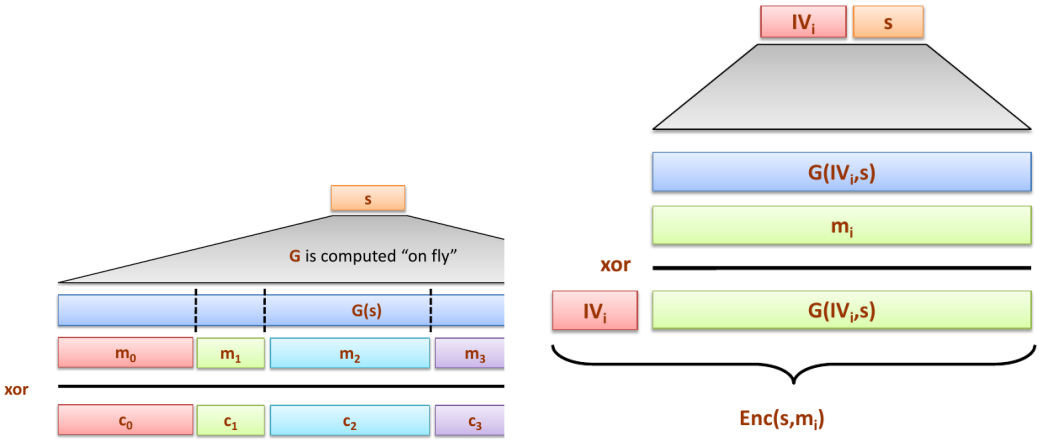

Figure 9: Synchronised (left) and unsynchronised mode (right)

RC4 Old but once widely used stream cipher. No security intuition, no proof! Problems:

1. No IV per default. Implementation specific: a) WEP uses an IV but too short (repeats after a day). b) Others reset IV on each restart...
2. Some output bytes are biased.
3. First bytes of output can leak information about the key.
4. Too short key lengths in some implementations (e.g. WEP) $\implies$ brute-forceable

ChaCha, Salsa Modern stream ciphers. Fast in software (as opposed to HW-accelerated AES). Addition/Rotation/XOR (ARX) make linear and differential cryptanalysis difficult. Combination in step 3 (see figure 10) gives hard-to-inverseability.

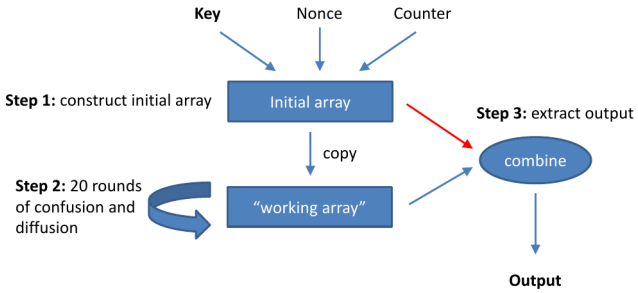

Figure 10: ChaCha schematics

4. Hash Functions and MACs

4.1. Message Authentication

Integrity How can Bob be sure that m really comes from Alice and has not been tampered with?

Message Authentication Code (MAC) scheme is a pair $(\text{Tag}, \text{Vrfy})$ where

$\text{Tag} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ is a tagging algorithm

$\text{Vrfy} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{yes, no\}$ is a verification algorithm.

Correctness: $\text{Vrfy}_k(m, \text{Tag}_k(m)) = yes$ should always hold.

Remark MACs per se do not protect against **replay attacks**. A simple workaround would be to include a timestamp in the message.

MAC security A given MAC is secure, if

$$\forall \text{ probabilistic poly-time adversaries } \mathcal{A} : \ P[\mathcal{A} \text{ breaks } MAC] = \varepsilon(n)$$

i.e. breaks it with probability negligible in the key size n. *Breaks* mean $\mathcal{A}$ produces a valid (m', t') pair for an unseen message m'.

Construction of a secure MAC using block ciphers:

For one block: $\text{Tag}_k(m) = F_k(m)$ (provable secure)

Generalising to multiple blocks:

1. Prepend counter to every block (to avoid permutation by adversary)
2. Include message length in every block (to avoid truncation)
3. Add a (per message) fresh random value to each block (to avoid splicing of different same-length messages)

Using all these measures results in a provable secure MAC. However, this construction is not practical. One Problem is that the tag is 4 times longer than the message, we need 4 times as many invocations of F_k.

CBC-MAC see figure 11

If we would not prepend $|m|$ then the adversary could forge $(m', t') = (m_1 || m_2 \oplus t_1, t_2)$ (after learning (m_1, t_1) and (m_2, t_2) in a training phase). This is known as a *splicing attack*.

Disadvantages of block-cipher-based MAC compared to hash functions-based MACs:

- Less efficient
- Many protected by export regulations

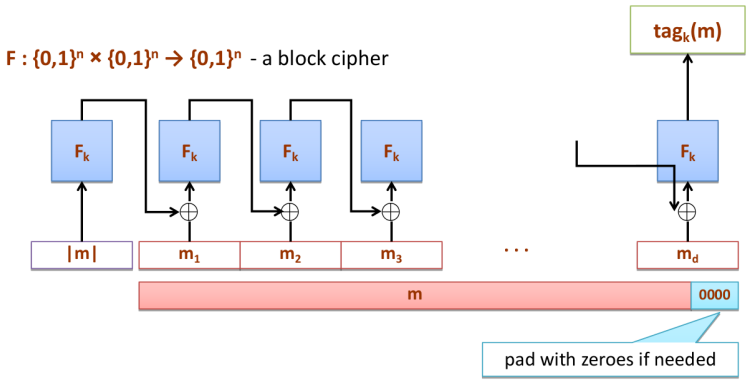

Figure 11: CBC-MAC

4.2. Hash functions

The basic property required for a hash function is collision resistance. Collisions always exist since the domain is much larger than the range the function maps to.

Collision resistance It should be "hard" to find a pair (m, m') such that $H(m) = H(m')$.

Security definition H is a collision-resistant hash function, if

$$\forall \text{ poly-time adversary } \mathcal{A} : \Pr[\mathcal{A} \text{ breaks } H] = \varepsilon(n)$$

The oracle selects s u.a.r., A outputs (m, m'). We say that adversary A breaks H, if $H^s(m) = H^s(m')$.

Problem: for fixed H there always exists an algorithm that finds a collision in constant time, because the hashing algorithm is fully known. Workaround: we use a key s.

Hash function A hash function is a probabilistic polynomial-time algorithm H such that:

H takes as input a key $s \in \{0,1\}^n$ and a message $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{L(n)}$ where $L(n)$ is some fixed function.[11]

Hash-based MAC A key for MAC is a pair (s, k). s is a key for hash function H and k for PRP F.

$$\text{Tag}((s,k), m) = F_k(H^s(m))$$

Theorem: If H and F are secure, then Tag is secure (proof by reduction).

Constructing hash functions

1. Construct a smaller fixed-input-length collision-resistant hash function h (aka. the *compression function*).
$$h : \{0,1\}^{2L} \to \{0,1\}^L$$
2. Use h to construct a variable length hash function H.
$$H : \{0,1\}^* \to \{0,1\}^L$$

[11] In reality, a hash function only takes a message as an input. The key only denotes the concrete instantiation out of the *family* of hash functions H^s.

Constructing a compression function Common approaches:

- block cipher
- specially designed function, e.g. SHA
- use Feistel Network as basis

Merkle-Damgard transform see figure 12. Note that $|m| = t$, $|m_i| = |IV| = |L|$. We append $m_{B+1} \coloneqq t$ to prevent padding attacks.

Theorem: If h is a collision-resistant compression function, then H is a collision-resistant hash function.

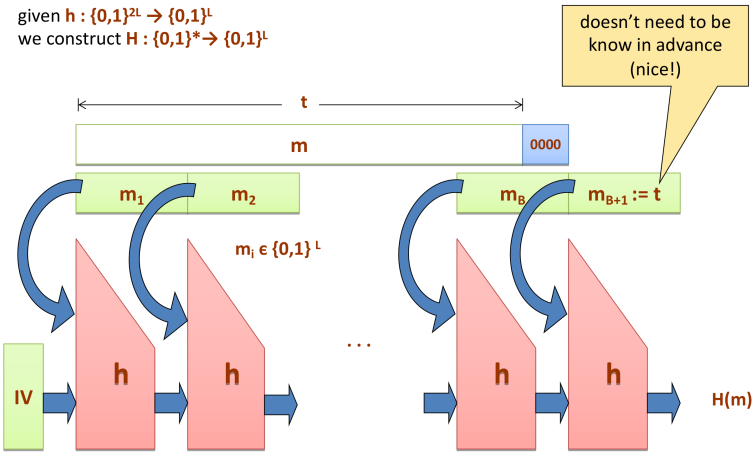

Figure 12: Merkle-Damgard construction

Concrete hash functions MD5 (broken), SHA-1 (broken), SHA-256, SHA-3 (does not use Merkle-Damgard, uses 'sponge construction')

Other Hash Function Properties Weaker security notions than collision resistance:

- Preimage resistance: Given hash value v, find x such that $h(x) = v$ ("inverting")
- 2^{nd} preimage resistance: Given x, find $x' \neq x$ such that $h(x) = h(x')$

Collision Resistance $\Rightarrow$ Preimage resistance

Collision Resistance $\Rightarrow$ 2^{nd} preimage resistance

Nested MAC (NMAC) see figure 13. Takes two keys k_1, k_2.

Disadvantages of NMAC

- Most real-world hash functions (MD5, SHA-1) do not permit to change IV (they use a fixed IV)
- The key (k_1, k_2) is too long

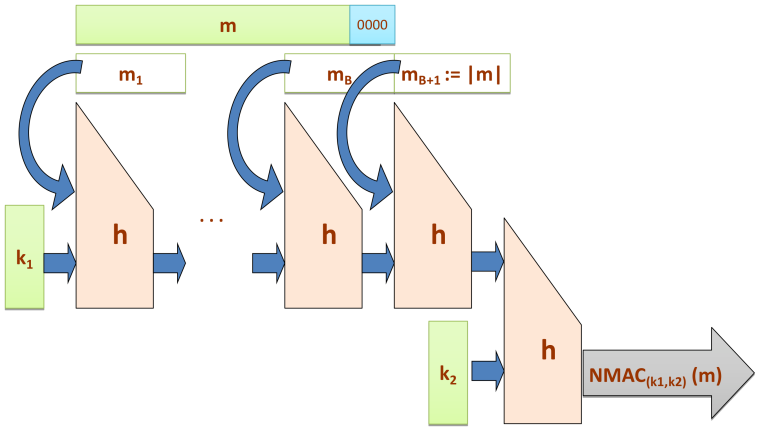

Figure 13: NMAC

Hash-based MAC (HMAC) see figure 14

Advantages of HMAC

- Easy to implement given H:

$$\mathrm{HMAC}_k(m) = H\big((k \oplus \mathrm{opad}) \,||\, H(k \oplus \mathrm{ipad} \,||\, m)\big)$$

- Has provable properties
- Is widely used in practice

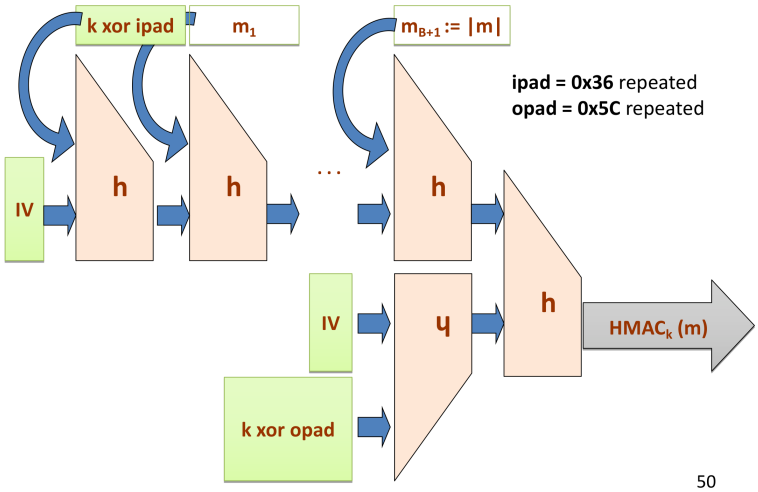

Figure 14: HMAC

Other uses Hash functions are used in practice to convert "non-uniform randomness" into a uniform one.

5. Public-Key Cryptography

The idea Instead of using one key k, we use two keys (pk, sk), where pk (public) is used for encryption and sk (private) is used for decryption.

Advantages of public keys: Simplified key management, i.e.

- secrecy not required
- fewer keys (Alice can use the same pair to communicate with multiple parties)

PKC for authentication sk is used for computing a tag (*signature*) and pk is used for verifying correctness of the tag.

Advantages of signature schemes:

- publicly verifiable[12]
- transferable
- provides non-repudiation (one cannot deny having signed m)

Trapdoor permutations A trapdoor permutation is a family of permutations indexed by $pk \in \mathcal{K}$;

$$\{\mathrm{Enc}_{pk} : \mathcal{M} \to \mathcal{C}\}_{pk \in \mathcal{K}}$$

such that for every key pk, there exists a key sk, and it is:

- easy to compute Enc_{pk}
- easy to compute $\mathrm{Enc}_{pk}^{-1} = \mathrm{Dec}_{sk}$ if one knows a trapdoor sk
- hard to compute $\mathrm{Enc}_{pk}^{-1} = \mathrm{Dec}_{sk}$ otherwise

PKC in practise: number theory We make use of mathematical problems that are known to be hard (hardness assumption). A suitable area of math is number theory.

Advantages of using number theory

- Security can (in principle) be based on famous mathematical conjectures.
- Constructions have a "mathematical structure". This allows us to create more advanced constructions (public key encryption, digital signature schemes, etc ...).
- Constructions have a natural security parameter: **size** (hence they can be "scaled").
- (A wonderful argument for theoreticians: a practical application of an area that was never believed to be practical.)

Disadvantages of using number theory

- Cryptography based on number theory is much less efficient.
- The number-theoretic "structure" can help the cryptanalysis.
- Certain problems that are believed to be hard for classical computers are shown to be easy for quantum computers.

[12] With symmetric MACs, Carol cannot verify that $(m, \mathrm{Tag}_k(m))$ comes from Alice and is only relayed by Bob. She cannot verify that $(m, \mathrm{Tag}_k(m))$ was not indeed created by Bob (given k is shared between Alice and Bob).

Cyclic groups (Easy way there) Let G be a group, $g \in G$, i be the order of g.

Then the cyclic group $\langle g \rangle = \{g^0, g^1, \ldots, g^{i-1}\}$ is a subgroup of G generated by g.

Notabene:

- Closed under multiplication: $g^a \cdot g^b = g^{a+b \bmod i}$
- Inverse always exist[13]: $(g^a)^{-1} = g^{i-a}$
- g^x can easily be computed using *square-and-multiply*

Discrete logarithms (Hard way back) Suppose G is cyclic, and g is its generator. For every element y, there exists x such that

$$y = g^x$$

Such a x will be called a *discrete logarithm* of y, and is denoted as $x \coloneqq \log y$.

Hardness assumption of the discrete logarithm In some groups[14], computing a discrete logarithm is believed to be hard. This includes:

- $Z_p^* = \{1, ..., p-1\}$ where p is prime
- groups based on elliptic curves

Note: $P = NP$ implies computing the discrete logarithm is easy.

Formal definition

Let H be an algorithm. Given input 1^n, H outputs a description of a group G of order q, such that $q = n$.

Oracle: Let (G, g) be the output of $H(1^n)$. The oracle selects a random element $y \leftarrow G$ from the group and returns (G, g, y) to the adversary.

A discrete logarithm problem is hard with respect to H, if

$$\forall \text{ probabilistic poly-time adversary } \mathcal{A} : \Pr[\mathcal{A} \text{ outputs } x \text{ s.t. } y = g^x)] = \varepsilon(n)$$

5.1. Diffie-Hellman key exchange

Key exchange Protocol through which two parties that initially share nothing can establish a shared secret despite the presence of an intermediate adversary.

Diffie-Hellman key exchange Let G be a group where discrete log is believed to be hard, let $q = |G|$, and let g be a generator of G. See figure 15 for the key exchange protocol.

[13] Proof left as an exercise.

[14] Groups of numbers, not groups of mathematicians!

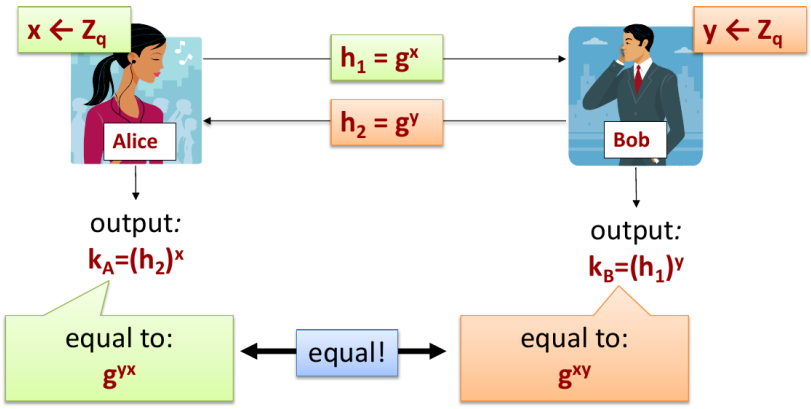

Figure 15: Diffie-Hellman key exchange

Security of a key exchange protocol

Informally: (Alice, Bob) is secure if no adversary can distinguish k from a random sequence r, given the transcript T of sent messages, with a non-negligible advantage.

Formally: We say (Alice, Bob) is a secure key exchange protocol if: the output (key) of Alice and Bob is always the same, and

$$\forall \text{ probabilistic poly-time adversary } \mathcal{A} : |\Pr[\mathcal{A}(1^n, k) = 1] - \Pr[\mathcal{A}(1^n, r) = 1]| = \varepsilon(n)$$

Is DH secure?

A) If the discrete log in G is easy, then the DH key exchange is <u>not</u> secure.

B) If the discrete log in G is hard, then it may also be insecure.

Quadratic residue QR $y \in G$ is a *quadratic residue* if there exists an $a \in G$ such that $a^2 = y$.

↯ Testing whether $y \in Z_p^*$ is a QR is possible in polynomial time.
$\Longrightarrow$ An adversary observing $y = g^x$ can learn whether x is even (iff y is QR) or odd, i.e. information about the parity of k is leaked: $k = g^{xy}$ is QR iff any of g^x and g^y is QR.
$\Longrightarrow$ Diffie-Hellman key exchange is <u>not</u> secure in Z_p^*!

Decisional Diffie-Hellman assumption

$$\forall \text{ PPT adversary } \mathcal{A} : |\Pr[\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(G, g, q, g^x, g^y, g^r) = 1]| = \varepsilon(n)$$

where r is random.

This does <u>not</u> hold in Z_p^* but it is believed to be hard in QR_p (which is a subgroup of Z_p^* such that p is a safe prime).

Safe prime p is a *safe prime* if $p = 2q + 1$, where q is also prime.

Man-in-the-middle attack Adversary that is not passively eavesdropping on the messages but is actively intercepting and manipulating them. Diffie-Hellman is <u>not</u> secure against it.

5.2. ElGamal encryption scheme

ElGamal encryption Encryption scheme which for each message runs a Diffie-Hellman key exchange to generate a new shared key. This key is then used to encrypt a single message. See figure 16.

Security of ElGamal If DDH is hard relative to G, then ElGamal is CPA-secure (under a passive adversary!).

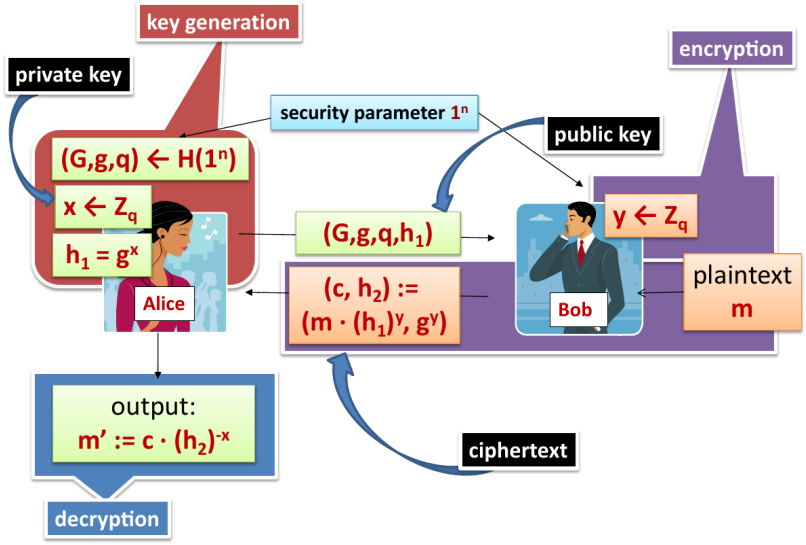

Figure 16: ElGamal encryption scheme

5.3. RSA encryption scheme

RSA key generation Given a security parameter 1^n:

1. Choose two prime numbers p and q of size n
2. Compute $N = pq$
3. Compute totient[15] $\Phi(N) = (p-1)(q-1)$
4. Choose e relative prime to $\Phi(N)$, i.e. $\gcd(e, \Phi(N)) = 1$
5. Compute $d = e^{-1} \mod \Phi(N)$ (i.e. d is multiplicative inverse of e, modulo $\Phi(N)$)

This yields a public key $pk = (e, N)$ and a private key $sk = (d, N)$. Note that p, q and $\Phi(N)$ remain secret.

RSA encryption & decryption

encryption:

$$c := m^e \mod N$$

decryption:

$$m := c^d \mod N$$

[15] Euler's totient function $\Phi(N)$ gives the number of integers smaller than N that are relatively prime to N. Recall that two numbers a, b are relatively prime if $\gcd(a, b) = 1$.

RSA encryption security Intuition: If attacker could factor N to p and q then she could:

- first compute $\Phi(N) = (p-1)(q-1)$ and
- then find the private key $d = e^{-1} \mod \Phi(N)$

$\Rightarrow$ Computing $\Phi(N)$ cannot be more difficult than factoring

Claim: computing $\Phi(N)$ is as hard as factoring N.

Proof outline: Suppose we can compute $\Phi(N)$. Then we know that $(p-1)(q-1) = \Phi(N)$ and $pq = N$. This is a system of 2 equations with 2 unknowns. It can easily be solved analytically for p and p.

RSA assumption It is hard to compute the e^{th} root (of c) without knowing $\Phi(N)$.

Or formally:

$$\forall \text{ PPT algorithm } \mathcal{A} : \Pr\left[m^e = c \mod N \mid m = \mathcal{A}(c, N, e)\right] = \varepsilon(|N|) = \varepsilon(k)$$

where $N = pq$ with p and q random primes and $|p| = |q| = k$, c is a random element of Z_N^*, and e is random element of $Z_{\Phi(N)}^*$ with $\gcd(e, \Phi(N)) = 1$.

CPA security in a public-key cryptosystem The is no need for a training phase with the oracle – knowing pk, the adversary can do the training herself.

Issues with "textbook" RSA

- **NOT CPA-secure:** No deterministic public key encryption scheme is! The adversary can simply do the encryption of the messages m_0, m_1 (which she gives to the oracle) herself and compare c_0 and c_1 to the c she got back from the oracle. Thus she can output b correctly with $p = 1$.
- **Homomorphic:**

  $$RSA_{e,N}(m_0 \cdot m_1) = (m_0 \cdot m_1)^e = m_0^e \cdot m_1^e = RSA_{e,N}(m_0) \cdot RSA_{e,N}(m_1)$$

  Using this the adversary can find out whether the to-her-unknown m, m_0, m_1 are related through $m_0 \cdot m_1 = m$ by checking whether $c_0 \cdot c_1 = c$. Thus we leak information.

To counter these issues, we usually *encode* the message prior to encryption with an encoding that adds randomness.[16] This makes the encryption non-deterministic and breaks RSA's algebraic properties.

With such encoding, RSA is believed to be CPA-secure. It is not, however chosen-ciphertext-secure.

Hybrid encryption RSA is a block cipher. Thus for long messages, it is less efficient than private key encryption. Instead, we use public key encryption to encrypt a symmetric key to use for the follow-up private key encryption.

[16] PKCS#1 defines the format for RSA's encoding. See RFC 8017 for details.

5.4. RSA signatures

RSA signing + verification process

Signing:

$$\sigma := m^d \mod N \xrightarrow{send} (m, \sigma)$$

Verification:

$$m' := \sigma^e \mod N \xrightarrow{check} m' \stackrel{?}{=} m$$

Security of signatures A signature scheme is *unforgeable under adaptive chosen-message attack* if:

$$\forall \text{ PPT adversary } \mathcal{A} : \Pr[\mathcal{A} \text{ outputs } (m, \sigma) \text{ s.t. } \text{Verify}(pk, m, \sigma) = True] = \varepsilon(n)$$

where (pk, sk) where chosen according to the security parameter 1^n.

In other words, it is unforgeable if the adversary can output a previously unseen (message, signature) pair that can be successfully verified.

RSA signatures in practise "Textbook" RSA signatures are <u>not</u> unforgeable because of homomorphism:

1. Query signature σ_1 for m_1
2. Query signature σ_2 for $m_2 = \frac{m}{m_1} \mod N$
3. Output (m, σ) such that $\sigma = \sigma_1 \cdot \sigma_2 \mod N$

Proof (for any m):

$$\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot m_2^d)^e = m_1^{ed} \cdot m_2^{ed} = m_1 \cdot m_2 = m \mod N$$

<u>Solution:</u> Hashed RSA

Signing:

$$\sigma := H(m)^d \mod N \xrightarrow{send} (m, \sigma)$$

Verification:

$$m' := \sigma^e \mod N \xrightarrow{check} m' \stackrel{?}{=} H(m)$$

This is widely used. Note that under the assumption that H is only collision-resistant, there exists <u>no</u> proof that this scheme is unforgeable. Such a proof only exists under the assumption that H is a truly random function.

5.5. Zero Knowledge Proofs

Proving knowledge of a secret s without revealing s.

Interactive protocol consists of prover P and verifier V.

Prover P: must prove knowledge of the secret

Verifier V: Verifies P's statement

Properties of ZKPs

1. **Completeness:** If both P and V act honestly then the protocol succeeds.
2. **Soundness:** P can only convince V if she knows s ("if the statement is true").

   Assume P can convince V with non-negligible probability. Then there must exist a *knowledge extractor* algorithm that, given P as a subroutine, computes the secret.
3. **Zero knowledge:** The proof does not leak any information on s.

   That is, there exists a simulator that, on input the initial state of V, outputs a "fake" transcript of the protocol. The fake transcript is indistinguishable (either perfectly, statistically, or computationally) from a real one.

   In other words, if a protocol transcript can be simulated, then V does not learn anything that she did not know before.

   A proof is *honest verifier zero knowledge* if the zero knowledge property at least holds for verifiers that adhere to the protocol. There exist transformations from constant round honest verifier zero knowledge protocols into zero knowledge arbitrary verifier protocols.

Examples of ZKPs include the Schnorr identification protocol (interactive), the Fiat-Shamir heuristic derived from it (non-interactive) and the Fiat-Shamir proof (interactive).

Schnorr identification protocol proves the knowledge of the discrete log of a public t.

Public protocol parameters are: p, q prime such that q divides $p-1$ and g generator of an order-q subgroup of Z_p^*.

The proof of honest verifier zero knowledge proceeds backwards: a protocol simulator **randomly**[17] chooses c and y and from that computes an appropriate x, leading to a fake trace (x, c, y).
Proof of soundness sends two challenges c_1, c_2 to the subroutine P and computes s from y_1, y_2.

Fiat-Shamir proof proves the knowledge of the square root (of a public t) in the RSA group.

Public protocol parameters are: n s.t $n = pq$ for large but unknown primes p, q.

[17] It is important to choose them uniformly at random so that the fake transcript has the same distribution with an honest verifier.

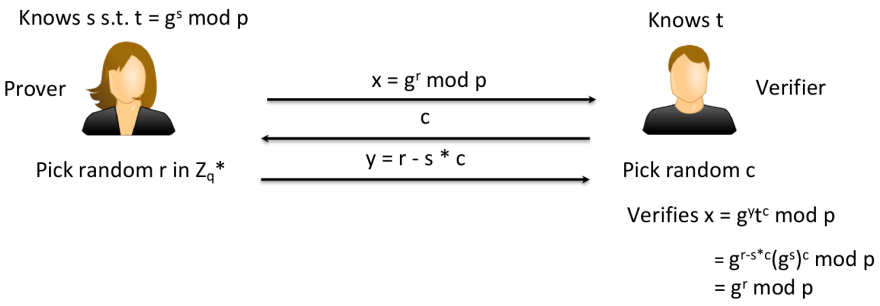

Figure 17: Schnorr identification protocol

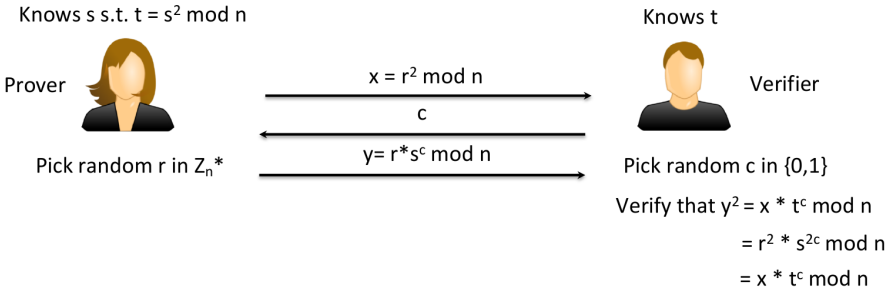

Figure 18: Fiat-Shamir proof

5.6. Commitment schemes

Commitment scheme Commit to a "hidden" value x that cannot be changed later.

1^{st} stage = commit: P "locks" x in a box and sends the box to V

2^{nd} stage = reveal: P gives the key to the box to V. V opens the box and learns x.

Properties of commitment schemes

Hiding: After the commit phase, V learns nothing about x

Binding: After the commit phase, there is only one value (i.e. x) that P can successfully reveal.

Exercise: Show that a commitment scheme can *not* be both perfectly hiding and perfectly binding at the same time.

Pedersen commitment scheme

- Setup (verifier)
  - Pick primes p and q such that q divides $p-1$
  - Pick generators g, h of the order-q subgroup of Z_p^* such that

$$a = \log_g(h) \Leftrightarrow h = g^a \mod p$$

  - Public parameters: p, q, g, h, Secret: a
- Commit (prover)

  - Pick random $r \in Z_q$
  - Output $c = g^x h^r \mod p$
- Reveal (prover)
  - Output x, r

Note that c is essentially a pseudo-random permutation of x.

Properties (intuition): Computationally binding because g^x is hard to invert (discrete log). Perfectly hiding comes from the randomness of h^r.

Part II.
Protocols and System Security

6. Introduction

6.1. Security and risk

Policies are formulated to achieve certain standard **security properties** (=security goals):

CIA

- **Confidentiality** No improper disclosure of information
- **Integrity** No improper modification of information
- **Availability** No improper impairment of functionality/service

Further properties

- Authenticity
- Non-repudiation (=accountability): one is responsible for actions
- Plausible deniability
- Privacy: in terms of anonymity and data protection
- Auditability: e.g. transaction logging

Unilateral security protects system against a single outside attacker.

Multilateral security protects multiple participants within the system from each other.

Risk minimisation Analysis should include: threat agents, threats, vulnerabilities, assets, countermeasures.

7. Security Protocols

7.1. Basic notions

Security protocol A *protocol* consists of a set of rules describing how messages are exchanged between principals acting in certain *roles*. It is a distributed algorithm with emphasis on security objectives.

A *security* (or *cryptographic*) *protocol* uses cryptographic mechanisms to achieve security objectives.

Standard symbolic attacker model (Dolev-Yao) An active attacker controlling the network:

- Can intercept and read all messages.
- Can decompose messages into their parts.
- Can construct and send new messages, any time.
- Can even compromise some agents and learn their keys.
- Can NOT break cryptographic primitives – i.e. decryption requires inverse keys

Protocol objectives

- **Entity authentication:** One party verifies the identity of a another party and that said party has recently and actively participated in the protocol. ("I am here now.")
- **Secrecy (Confidentiality):** Data available only to those authorized to obtain it.
  For keys, this is sometimes called **key authentication**.
- **Freshness:** Data is new, i.e. not replayed from an older session.
- **Key confirmation:** Assurance that the other second party actually possesses a given key.

Entity agreement A protocol guarantees that an agent A has **non-injective agreement** with B on a set of data items ds if:

Whenever A completes a run of the protocol in role R1 (apparently with B) then B has been running the protocol (apparently with A) and B was acting as R2 in his run, and both agree on ds.

See figure 19 for a visual representation representation of **injective agreement** where each run of A must correspond to a unique run of B.

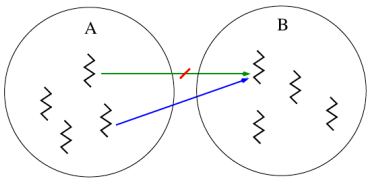

Figure 19: Injective agreement

7.2. Problems and principles

We aim to build a key establishment protocol from first principles. We have the following roles: an *initiator* A, a *responder* B and an *honest server* S.

Honest server Never cheats and never reveals user secrets. Sometimes (wrongly) called a "trusted server" – but honesty $\neq$ trust!

Adversary model There are different threat assumptions. 1)-3) are a worst-case network adversary (cf Dolev-Yao). The adversary can:

1) eavesdrop on messages, but cannot break cryptography
2) completely control the network, i.e.
   - immediately intercept, modify, and fake messages
   - compose/decompose messages with the available keys
3) participate in the protocol (as insider or outsider)
4) obtain old session keys

Types of attacks

- **Man-in-the-middle (MITM) (aka parallel sessions)** $A \leftrightarrow M \leftrightarrow B$
- **Replay** record and later replay messages (or parts thereof) $\Rightarrow$ use challenge-response based on nonces
- **Reflection** send information back to the originator
- **Oracle** take advantage of normal protocol responses as encryption and decryption "services"
- **Type flaw** substitute a different type of message field (e.g. a key in place of an identifier). Works since messages are sent as a bit string without type information.
- **Guessing**
- **Eavesdropping** $\Rightarrow$ encrypt session keys using long-term keys
- **Binding** $\Rightarrow$ cryptographically bind names to session keys

Needham-Schroeder public-key protocol NSPK Example of a security protocol. Its goal is mutual authentication, i.e. establishing a shared secret.

$$1.\ A \to B:\ \{A, N_A\}_{K_B}$$
$$2.\ B \to A:\ \{N_A, N_B\}_{K_A}$$
$$3.\ A \to B:\ \{N_A B\}_{K_B}$$

Informally, correctness is meant to rise from B sending back N_A to A gives A assurance that B must be who she claims – only she can decrypt the initial message encrypted with K_B.

$\Rightarrow$ MITM attack: as shown in figure 20, A correctly believes that she is talking to C. However C is able to trick B into believing she is talking to A.

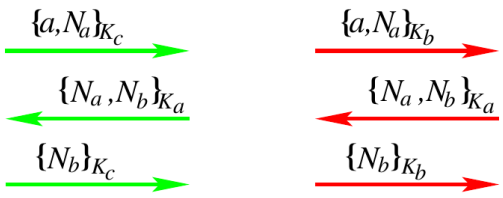

Figure 20: MITM attack on NSPK with actors $A \leftrightarrow C \leftrightarrow B$

Otway-Rees protocol Server-based, with key authentication and key freshness but without entity authentication or key confirmation.

Assumes K_{AS} and K_{BS} are pre-shared. I is some run identifier.

$$\text{M1. } A \to B: \ \{I, A, B, \{N_A, I, A, B\}_{K_{AS}}\}$$
$$\text{M2. } B \to S: \ \{I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}\}$$
$$\text{M3. } S \to B: \ \{I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}\}$$
$$\text{M4. } B \to A: \ \{I, \{N_A, K_{AB}\}_{K_{AS}}\}$$

$\Rightarrow$ reflection/type flaw attack:

Assume that $|\{I, A, B\}| = |\{K_{AB}\}|$. Then M can omit M2 and M3 and directly replay message M1 as M4:

$$\text{M1. } A \to M(B): \ \{I, A, B, \{N_A, I, A, B\}_{K_{AS}}\}$$
$$\text{M4. } M(B) \to A: \ \{I, \{N_A, I, A, B\}_{K_{AS}}\}$$

In a similar fashion, M can impersonate S:

$$\text{M1. } A \to B: \ \{I, A, B, \{N_A, I, A, B\}_{K_{AS}}\}$$
$$\text{M2. } B \to M(S): \ \{I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}\}$$
$$\text{M3. } M(S) \to B: \ \{I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}\}$$
$$\text{M4. } B \to A: \ \{I, \{N_A, I, A, B\}_{K_{AS}}\}$$

In both cases the honest agents accept a wrong session key $\{I, A, B\}$ known to M, thus violating key authentication/secrecy.

Similar attacks to the ones above exist on the Andrew Secure RPC protocol and the Denning-Sacco protocol (key exchange with a CA).

Freshness mechanisms

- Nonce: $\longrightarrow$ ↯ recipient must store previous values
- Counter: $\longrightarrow$ ↯ guessable, desynchronisation
- Timestamps: $\longrightarrow$ ↯ desynchronisation of time windows
- Challenge reponse with nonce $\longrightarrow$ ↯ more messages

7.3. Introduction to formal models

Trace a sequence of events. May interleave between different protocol runs.

Formal symbolic model M of a protocol. *Formal* meaning having well-defined mathematical semantics, and *symbolic* meaning we abstract away bit-strings to algebraic terms. The model will be a transition systems describing all agent actions.

The semantics of a security procol P is a set of traces:

$$\|P\| = traces(P)$$

The security goal/property ϕ also denotes a set of traces: $\|\phi\|$

P *satisfies* ϕ (i.e. $P \models \phi$) iff

$$\|P\| \subseteq \|\phi\|$$

Attack traces are those in

$$\|P\| - \|\phi\|$$

Protocol analysis is done using **model-checking**. Tools employ **state enumeration** to find an attack in the tree of possible states.

We formalise the analysis power of a Dolev-Yao adversary as follows:

Let T be the initial set of messages known to the adversary. $has(T)$ is the smallest set of messages inferable from T, using the following derivation rules:

- $elem:$ $\quad t \in T \implies t \in has(T)$
- $pair:$ $\quad t_1 \in has(T) \wedge t_2 \in has(T) \implies (t_1, t_2) \in has(T)$
- $fst, snd:$ $\quad (t_1, t_2) \in has(T) \implies t_1 \in has(T) \wedge t_2 \in has(T)$
- $enc:$ $\quad t_1 \in has(T) \wedge t_2 \in has(T) \implies \{t_1\}_{t_2} \in has(T)$
- $hash:$ $\quad t_1 \in has(T) \implies hash(t_1) \in has(T)$
- $dec:$ $\quad \{t_1\}_{t_2} \in has(T) \wedge t_2^{-1} \in has(T) \implies t_1 \in has(T)$

7.4. Modeling using multiset rewriting (MSR)

Multiset A *multiset* m over a set X is a set of elements, each imbued with a multiplicity, i.e. $m : X \mapsto \mathbb{N}$, where $m(x)$ denotes the multiplicity of x.

We use $\subseteq^\sharp$ for multiset inclusion, $\cup^\sharp$ for multiset union, and $\setminus^\sharp$ for multiset difference. We denote by $X^\sharp$ the set of finite multisets with elements from X.

e.g. $m = [0 \mapsto 1, 1 \mapsto 3, 2 \mapsto 2] = [0, 1, 1, 1, 2, 2]$

Fact Let Σ_{fact} be a set of untyped *fact symbols*, each with an arity $k \geq 0$. Then

$$F(t_1, \ldots, t_k)$$

for $F \in \Sigma_{\text{fact}}$ with arity k and $(t_1, \ldots, t_k) \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$ is called a *fact*. Here we also have terms t_i, a set of variables $\mathcal{X}$ and a set of names $\mathcal{N}$.

Outlook: We distinguish between **linear** facts and **persistent** facts: the former are "consumed" by a rewriting step (i.e. removed from the state) while the latter are not (and can thus be reused arbitrarily often).

Labeled multiset rewriting A *labeled multiset rewriting rule* is a triple

$$l \xrightarrow{a} r$$

where l, r and a are all multisets of facts. l and r are called *state facts* and a is called *action facts* or *events*.

A *labeled multiset rewriting system* is a set of labeled multiset rewriting rules.

The are four different types of rules that we use for modeling:

Adversary rules Determine which messages the adversary can derive from her knowledge. The adversary fact $K(t)$ represents the knowledge of a term t.[18]

The adversary can use the following inference rules:

$$\frac{\text{Out}(x)}{K(x)} \qquad \frac{K(x)}{\text{In}(x)}\ K(x) \qquad \frac{\text{Fr}(x)}{K(x)} \qquad \frac{K(t_1) \ldots K(t_k)}{K(f(t_1, \ldots, t_k))}\ f \in \Sigma(k\text{-ary})$$

(1) Out facts mark messages sent by the protocol

(2) In facts mark messages to be received by the protocol[19]

(3) The adversary can use fresh values

(4) The adversary has inference capabilities

A protocol model may include additional adversary rules, e.g. for compromising other agents by learning their long-term keys.

[18] c.f. *has* in section 7.3

[19] Note that the top $K(x)$ is a state fact while the right $K(x)$ is an action fact.

Protocol rules Formalise the roles of the protocol under study. Define the sending and receiving of messages and use agent state facts to keep track of the role's progress.

A protocol consists of *roles*. Each role consists of a set of protocol rules that specify the sending and receiving of messages and the use of fresh values. Roles use *agent state facts* to keep track of their progress.

An **agent state fact** for role R is fact

$$\text{St\_R\_s}(A, id, k_1, ..., k_n)$$

where $\text{St\_R\_s} \in \Sigma_{fact}$ and

- $\text{s} \in \mathbb{N}$ is the number of the protocol step within the role
- A is the name of the agent executing the role
- id is the thread identifier for this instantiation of role R
- $k_i \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$ are terms in the agent's knowledge

As for communication: messages are sent and received via Out and In (state) facts. Any rule with such a fact also has a matching $Send$ and $Recv$ action fact.

e.g. $[\text{St\_I\_2}(A, id, k), \text{In}(m)] \xrightarrow{\text{Recv}(A,m)} [\text{St\_I\_3}(A, id, k, m)]$

Infrastructure rules Formalise the generation of cryptographic keys, e.g. to model a public-key infrastructure (PKI).

Rule for generating long-term private and public keys:

$$[\text{Fr}(sk)] \longrightarrow [\text{Ltk}(A, sk), \text{Pk}(A, pk(sk)), \text{Out}(pk(sk))]$$

- requires a fresh value sk
- fact $\text{Ltk}(A, sk)$ records sk as A's long-term private key
- fact $\text{Pk}(A, pk(sk))$ records $pk(sk)$ as A's long-term public key
- fact $\text{Out}(pk(sk)$ publishes the public key $pk(sk)$.

Fresh rule Generates unique fresh values X marked as fresh facts $\text{Fr}(X)$. These can be used as nonces or thread identifiers.

We have a set of *fresh values* $FV \subseteq \mathcal{N}$ and a set of *public values* $PV \subseteq \mathcal{N}$. Both are countably infinite but disjoint: $FV \cap PV = \emptyset$. We use terms in $\mathcal{T}_\Sigma(\mathcal{X}, \mathcal{N})$.

The fresh rule is defined as follows:

$$[\,] \longrightarrow [\text{Fr}(N)]$$

It has no precondition and each created nonce N is unique.

Initialization For each role R there must be an initialization rule which creates a thread with a fresh identifier id, playing role R and owned by an agent A:

$$\big[\text{Fr}(id), \text{Ltk}(A, skA), \text{Pk}(B, pkB)\big] \xrightarrow{\text{Create\_R(A, id)}} \big[\text{St\_R\_1}(A, id, skA, B, pkB), \text{Ltk}(A, skA), \text{Pk}(B, pkB)\big]$$

Ground Instances

- An *instance* X_σ of an object X (term, fact, rewrite rule,...) is the result of applying a substitution σ to all terms in X
- A *ground instance* of X is an instance where all resulting terms have no more variables ("are ground")
- $ginsts(R)$ denotes the set of all ground instances of rules in a multiset rewrite system R
- $\mathcal{G}$ is the set of ground facts. $\mathcal{G}^\sharp$ then is the set of finite multisets of elements from $\mathcal{G}$

State is a finite multiset of ground facts (i.e an element of $\mathcal{G}^\sharp$)

Rewriting step The labeled transition relation $steps(R) \subseteq \mathcal{G}^\sharp \times ginsts(R) \times \mathcal{G}^\sharp$ in a multiset rewrite system R is defined as follows:

$$\frac{I \xrightarrow{a} r \in ginsts(R) \qquad I \subseteq^\sharp S \qquad S' = (S \setminus^\sharp I) \cup^\sharp r}{(S, I \xrightarrow{a} r, S') \in steps(R)}$$

Note: This only applies to linear facts, which are consumed by the rewriting step (i.e. removed from the state). For the general case (with persistent facts) the rule is slightly different.

Execution An execution of R is an alternating sequence $S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow{a_k} r_k), S_k$ of states and rewrite rule instances such that:

- the initial state is empty: $S_0 = \emptyset^\sharp$
- it is a transition sequence: $\forall i.\ (S_{i-1}, l_i \xrightarrow{a_i} r_i, S_i) \in steps(R)$
- fresh names are unique: $\forall n, i, j.\ (l_i \xrightarrow{a_i} r_i) = (l_j \xrightarrow{a_j} r_j) = ([\,] \to [\mathrm{Fr}(n)]) \implies i = j$

Trace The trace tr of an execution $S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow{a_k} r_k), S_k$ is defined by the sequence of the multisets of its action lables, i.e. by

$$a_1, a_2, ..., a_k$$

Events occur in traces. Properties of traces are expressed in terms of events. Common events are:

- Protocol events: $\mathrm{Send}(A, t)$, $\mathrm{Recv}(A, t)$, $\mathrm{Create\_R}(A, id)$
- Claim events: $\mathrm{Claim\_}\texttt{claimtype}(A, t)$
- Honesty and reveal events: $\mathrm{Honest}(A)$, $\mathrm{Rev}(A)$
- Adversary knowledge: $\mathrm{K}(t)$

Claim events are specific action facts. Their only effect is to record facts or claims in the protocol trace since they cannot be observed, modified or generated by the adversary.

Often properties are formulated from the point of view of a given role R by including the role identifier R in the claim, thus yielding guarantees for that role.

An event F can be timestamped: $F@i$ holds on trace $tr = a_1, \ldots, a_n$ if $F \in a_i$.

7.5. Formalising security properties

7.5.1. Secrecy

Compromised agent An agent A under full adversary control, i.e. sharing its long-term secrets with the adversary. Modeled through the following rewrite rule:

$$\big[\mathrm{Ltk}(A, skA)\big] \xrightarrow{\mathrm{Rev}(A)} \big[\mathrm{Ltk}(A, skA), \mathrm{Out}(skA)\big]$$

Honesty An agent A is honest in a trace tr if:

$$\mathrm{Rev}(A) \notin tr$$

Claim events are usually accompanied by $Honest(B)$ events specifying the owner and the intended communication partner(s).

Secrecy A trace tr satisfies the secrecy property if:

$$\forall A, M, i.\quad \big(\mathrm{Claim\_}\texttt{secret}(A, M)@i\big) \implies \neg\big(\exists j.\ K(M)@j\big) \vee \Big(\exists B, k.\ \mathrm{Rev}(B)@k \wedge \mathrm{Honest}(B)@i\Big)$$

i.e. either the adversary does not know M or (if he does) it must have been revealed by an agent that is required to be honest.

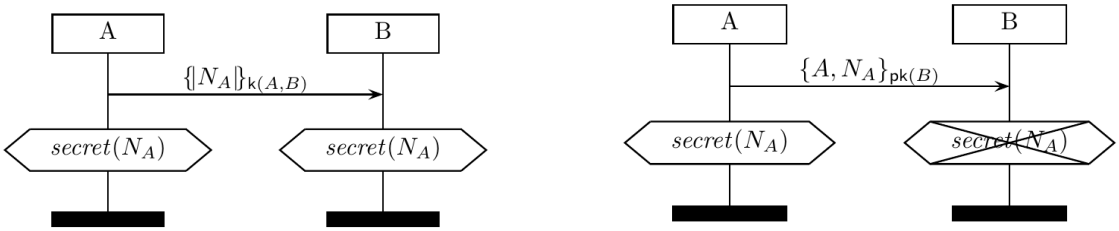

Figure 21: Message sequence charts where secrecy holds (left) and does not hold (right)

7.5.2. Authentication

We use for following four (increasingly stronger) authentication properties, adapted from Gavin Lowe, that allow role R1 to authenticate role R2:

Aliveness A protocol guarantees to an agent a in role R1 *aliveness* of another agent b if, whenever a completes a run of the protocol, apparently with b in role R2, then b has previously been running the protocol.

Weak agreement A protocol guarantees to an agent a in role R1 *weak agreement* with another agent b if, whenever agent a completes a run of the protocol, apparently with b in role R2, then b has previously been running the protocol, apparently with a.

Non-injective agreement A protocol guarantees to an agent a in role R1 *non-injective agreement* with an agent b in role R2 on a message M if, whenever a completes a run of the protocol, apparently with b in role R2, then b has previously been running the protocol, apparently with a, and b was acting in role R2 in her run, and the two principals agreed on M.

Injective agreement is non-injective agreement where additionally each run of agent a in role R1 corresponds to a unique run of agent b. That is, no other run of a matches this run of b.

To formalise these, we introduce two claim events:

- Claim_`commit`$(R1, R2, t)$ – at the end of role R1, when she can construct t
- Claim_`running`$(R2, R1, u)$ – after R2 can construct u (her view of t) and causally preceding R1's commit claim.

Non-injective agreement (formally) The property $Agreement_{NI}(R1, R2, t)$ consists of all traces satisfying:

$$\forall a, b, t, i.\ \text{Claim\_}\texttt{commit}(a, b, \langle R1, R2, t\rangle)@i$$
$$\implies (\exists j.\ \text{Claim\_}\texttt{running}(b, a, \langle R1, R2, t\rangle) \vee (\exists X, r.\ \text{Rev}(X)@r \wedge \text{Honest}(X)@i)$$

Note that is not necessary to explicitly ask for $j < i$. This is because it must hold for all traces, so we can always consider some trace prefix that ends with the commit claim.

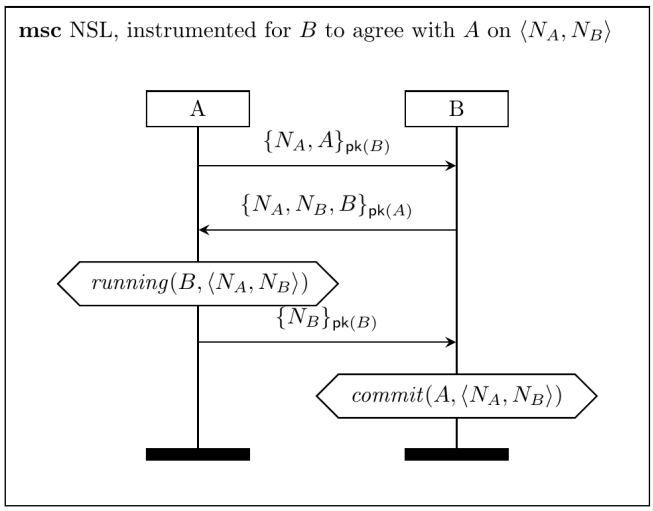

Figure 22: Message sequence chart of a protocol with non-injective agreement.
Note that claims are again shortened, e.g. from Claim_`commit`$(B, A, \langle A, B, N_A, N_B\rangle)$

Injective agreement (formally) The property $Agreement_I(R1, R2, t)$ consists of all traces satisfying:

$$\forall a, b, t, i.\ \text{Claim\_}\texttt{commit}(a, b, \langle R1, R2, t\rangle)@i$$
$$\implies \Big(\exists j.\ \text{Claim\_}\texttt{running}(b, a, \langle R1, R2, t\rangle)@j \wedge \neg(\exists a_2, b_2, i_2.\ \text{Claim\_}\texttt{commit}(a_2, b_2, \langle R1, R2, t\rangle)@i_2 \wedge \neg(i_2 = i))\Big)$$
$$\vee(\exists X, r.\ \text{Rev}(X)@r \wedge \text{Honest}(X)@i)$$

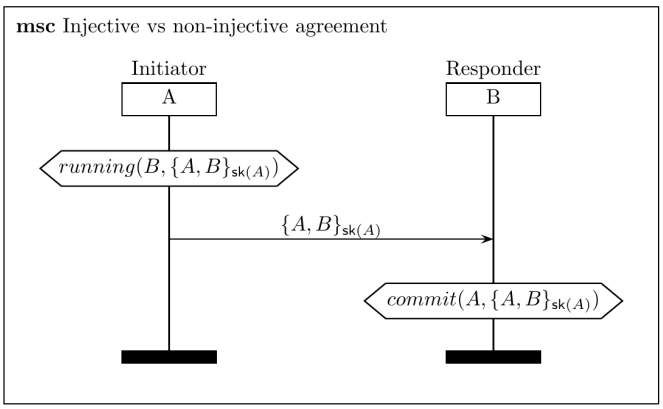

Figure 23: MSC where non-injective agreement holds but injective agreement fails because an adversary can replay the message to several threads in responder role B

Weak agreement (formally) A trace tr satisfies the property $Agreement_{weak}$ iff:

$$\forall a, b, i.\ \text{Claim\_}\texttt{commit}(a, b, \langle\rangle)@i$$
$$\implies (\exists j.\ \text{Claim\_}\texttt{running}(b, a, \langle\rangle)@j) \vee (\exists X, r.\ \text{Rev}(X)@r \wedge \text{Honest}(X)@i)$$

That is, the agent agree that they are communicating. No statement is made about their roles and the data to agree on.

Aliveness (formally) A trace tr satisfies the property $Alive$ iff:

$$\forall a, b, i.\ \text{Claim\_}\texttt{commit}(a, b, \langle\rangle)@i$$
$$\implies (\exists id, R, j.\ \text{Create}(b, id, R)@j) \vee (\exists X, r.\ \text{Rev}(X)@r \wedge \text{Honest}(X)@i)$$

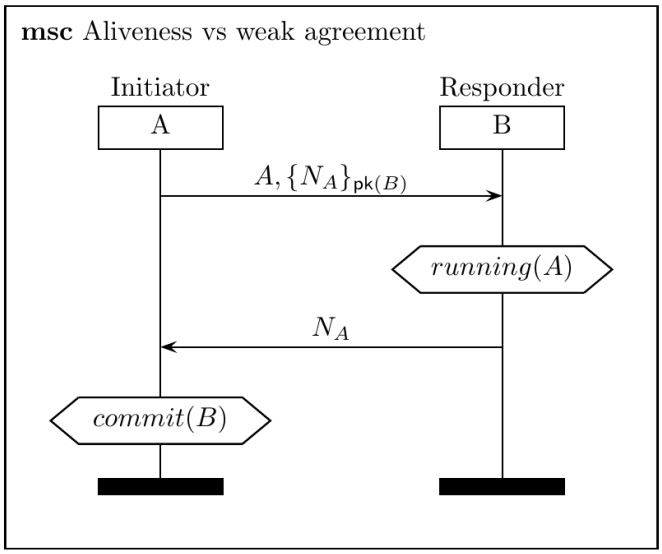

Figure 24: MSC where aliveness holds but weak agreement fails because an adversary can modify unprotected A to C

Summary From alive to injective agreement – the property guarantees to agent a executing the protocol (apparently with agent b) that...

...and run of b is unique
...in role R2 and agreed on message M...
...thinking with a...
...b ran protocol...

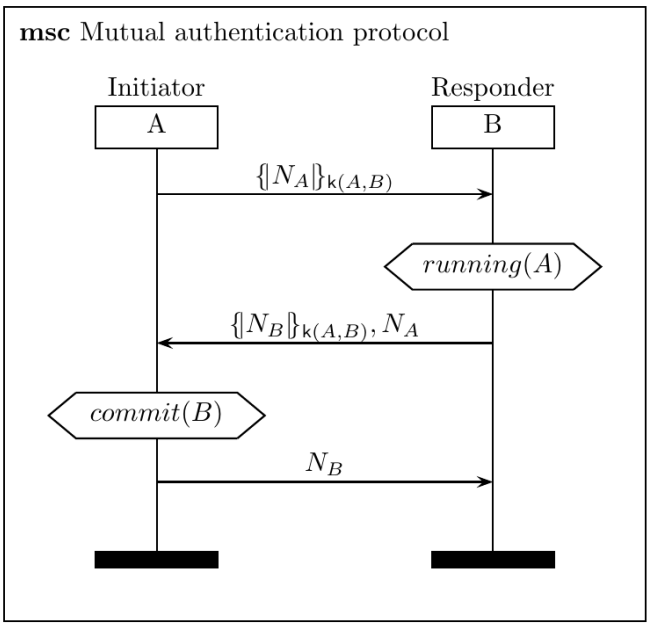

Figure 25: MSC where aliveness fails because an adversary can reflect messages causing A to run the protocol with herself

8. Network Security

8.1. Networking recap

Recall that:

- Network stacks are *layered*. This entails the need for *encapsulation* of packets.
- Neither IP nor TCP provide authenticity or confidentiality. Addresses can be faked, payloads can be modified.

How do we secure the network? Inbetween where – end-to-end, hop-to-hop, gateway-to-gateway? At which layer?[20]

Application
Transport
Network
Link
Physical

Table 2: Layered network stack

8.2. Application-managed security

Let the application be responsible for ensuring the security of the communication, following the *end-to-end principle*.

[20] Exercise: For the following protocols, explain which layer they work at and which sections of the network they protect: PGP, Signal Protocol, TLS, SMTP, WPA, IPsec.

- ⊕ Stronger guarantees than when on lower level (e.g. application checksum on data versus TCP checksum)
- ⊕ No need to trust network and its nodes
- ⊕ Security decision can be based on user data
- ⊖ Implementation and design errors (bugs, sidechannel attacks, bad crypto, ...)
- ⊖ Users cannot be assumed to be perfect (misunderstanding, malice, social engineering, ...)
- ⊖ Need network mechanism anyway (e.g. to lock down network during incident response)
- ⊖ Interference with middleboxes (email virus scanning or archiving, ...)
- ⊖ What are the endpoints? (users, devices or servers) $\implies$ trust-to-trust instead of end-to-end

8.3. Network-managed security

Let some other level of the network stack ensure the security of the communication.

- ⊕ Implement once (e.g. in OS) rather than in each application
- ⊕ No burden on the users (security is transparent to them)
- ⊕ Adds security to applications that cannot be modified (enterprise software ...)
- ⊕ Some only make sense as middleboxes because they need a global view of the traffic
- ⊖ No authentication of higher levels (e.g. IPsec authenticates IPs but not user)
- ⊖ WPA: only secures a single link (alas the most vulnerable)

8.4. Network security in practice

In general, there is a tradeoff between *defense-in-depth* versus *keep-things-simple*. Also, securing **anything** will always be an arms race.

Firewall Establishes a *trust perimeter*, and controls everything that crosses it (idea: easier to gain privileges on a host). Some types of firewalls are:

- *Packet filter*: Router performing access control, e.g. by only allowing traffic from certain IP addresses, ports or certain protocols. Easy to setup but very coarse control.
- *Application-layer proxy*: Inspects and filters traffic to/from an application (e.g. email virus scanning, SQL injection attacks).

Advantages and disadvantages of firewalls:

- ⊕ Scales better than host security
- ⊕ Secures insecure legacy systems
- ⊙ Single point of failure (thus single point we need to patch)
- ⊖ Relevant parts of packet/data may be encrypted
- ⊖ Hard-to-define trust perimeter (cloud services, rogue users)

⊖ Packet filter: does not handle HTTP and email traffic well[21]

Intrusion detection system Monitors and analyses system or network events for signs of possible incidents, which represent (imminent) violations of computer security policies. They can be implemented on a:

- *Host-level*: monitor events of individual hosts, e.g. to identify break-ins or insider attacks
- *Network-level*: examine network-wide traffic flow, e.g. to identify DDOS attacks

They can employ ... mechanisms:

- *Signature-based*: recognise patterns corresponding to known threats (e.g. log traces, root logins, ...). Only detects known threats.
- *Anomaly based*: use ML to classify "normal" behaviour, compare that with observed current events (e.g. number of sent mails, number of failed login attempts, process usage, ...). **May** detect unknown threats.

Advantages and disadvantages of intrusion detection systems:

⊕ Supports mitigation through earlier alerting

⊖ Limited effectiveness (ability of rules to capture future attacks)

⊖ False alarms (costly and eventually ignored)

Design principles Common patterns and guidelines for security engineering. For example:

- Trust-to-trust, end-to-end
- Keep it simple (KISS)
- Defense in depth
- Anything not explicitly required must be forbidden

[21] thgoebel: Why though? A packet filter can quite happily block traffic to/from untrusted/unwanted IP ranges. The only thing that HTTP/email traffic has in common (as opposed to other traffic) is the port!?

9. Security Protocols II

Goal: Learn underlying principles by examining four different standard protocols.

First we examine user authentication and authorisation.

- Centralised systems: easy! Design OS so that access requests pass through a gatekeeper that is authenticating users
- Distributed systems: more challenging
- Early days: Authentication by assertion, client authenticates user and server enforces security policy based on user ID ⇒ **Unsafe!** Attacker with network access can spoof user IDs
- Not better: sending passwords in clear, as in e.g. HTTP Basic Authentication

9.1. Kerberos

Three heads symbolise *authentication, authorization* and *audit*. The protocol realises the first two.

Single sign-on (SSO) protocol one password per session, subsequent authentication behind the scenes

Requirements

- **Secure:** Only authenticated users can access authorized resources.
- **Single Sign-On:** Single password needed to access network services, unawareness of underlying protocols.
- **Scalable:** Ability to scale with the number of users and servers (while remaining to be easy to administrate).
- **Available:** Since many services depend on Kerberos for access control, it must be reliable. Bottlenecks should also be avoided. (Realised by replicating the Kerberos server)

High-level idea Trusted server T distributes keys, thus creating a secure channel between A and B. Assumes initial shared key between each client and T. See Figure 26.

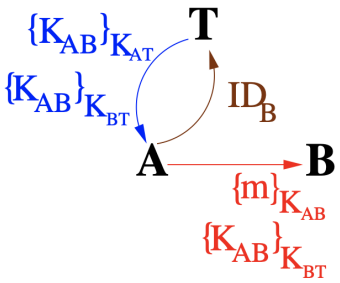

Figure 26: Setup of secure channel

Kerberos protocol is loosely based on the Nedham-Schroeder Shared-Key protocol:

M1. $A \to T : A, B, N_1$

M2. $T \to A : \{N_1, B, K_{AB}, \{K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$

M3. $A \to B : \{K_{AB}, A\}_{K_{BT}}\}$

M4. $B \to A : \{N_2\}_{K_{AB}}$

M5. $A \to B : \{N_2 - 1\}_{K_{AB}}$

The nonces ensure the *freshness* of key K_{AB}.

Note that A must reply with $\{N_2 - 1\}$ (i.e. it must slightly alter the nonce) in order to prevent replay attacks. For the same reason (i.e. prevent replay attacks by ensuring *recentness* of the session key K_{AB}), Kerberos uses timestamps instead of nonces. However this requires the clocks of the agents to be synchronised in order for authentication to succeed.

Architecture of Kerberos v4

- Authentication: Kerberos Authentication Server KAS (Message 1 and 2) – once per login session
- Authorization: Ticket Granting Server TGS (Message 3 and 4) – once per type of service
- Access Control: service server checks TGS tickets – once per service session

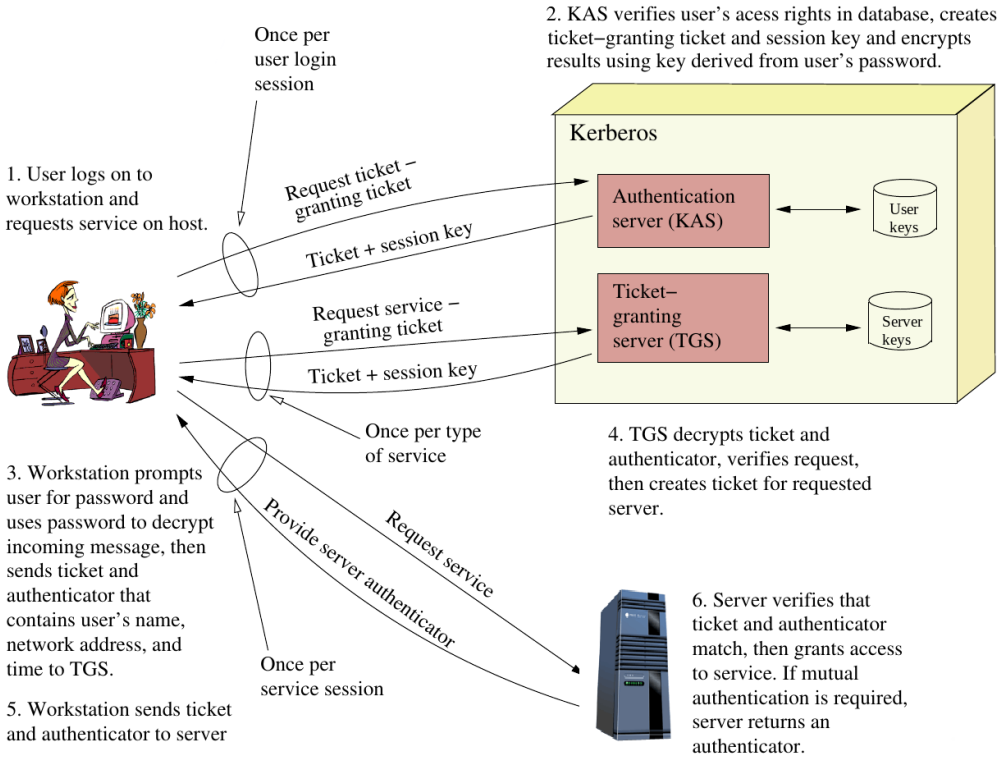

Figure 27: Kerberos message flow

Authentication phase Get an $AuthTicket$ from the KAS:

M1. $A \rightarrow KAS : A, TGS$

M2. $KAS \rightarrow A : \{\mathcal{T}_1, TGS, K_{A,TGS}, \underbrace{\{A, TGS, K_{A,TGS}, \mathcal{T}_1\}_{K_{KAS,TGS}}}_{AuthTicket}\}_{K_{A,KAS}}$

i) A logs onto workstation and requests access to network services (in general, not yet to a specific service)

ii) Precondition: user password and $K_{KAS,TGS}$ must be in the database

iii) KAS accesses database and replies with a session key $K_{A,TGS}$ (lifetime several hours) and an encrypted ticket $AuthTicket$. The reply is encrypted under $K_{A,KAS}$, which is derived from the user's password.

iv) A types password on workstation to decrypt results, which are stored for session. When $K_{A,TGS}$ expires A is logged out.

Authorisation phase Get a $ServTicket$ from the TGS:

M3. $A \rightarrow TGS : \underbrace{\{A, TGS, K_{A,TGS}, \mathcal{T}_1\}_{K_{KAS,TGS}}}_{AuthTicket}, \underbrace{\{A, \mathcal{T}_2\}_{K_{A,TGS}}}_{authenticator}, B$

M4. $TGS \rightarrow A : \{K_{AB}, B, \mathcal{T}_3, \underbrace{\{A, B, K_{AB}, \mathcal{T}_3\}_{K_{B,TGS}}}_{ServTicket}\}_{K_{A,TGS}}$

i) A presents the $AuthTicket$ from M2 to the TGS together with a new $authenticator$ (lifetime few seconds).
Note that while tickets can be reused multiple times, authenticators cannot. Their short, time-bound validity prevents replay attacks.

ii) TGS issues A a new session key $K_{A,B}$ (lifetime few minutes) and a new ticket $ServTicket$ only readable by the network resource B (thus authorising A's access to B).

Service phase Access the network service:

M5. $A \rightarrow B : \underbrace{\{A, B, K_{AB}, \mathcal{T}_3\}_{K_{B,TGS}}}_{ServTicket}, \underbrace{\{A, \mathcal{T}_4\}_{K_{AB}}}_{authenticator}$

M6. $B \rightarrow A : \{\mathcal{T}_4 + 1\}_{K_{AB}}$

i) A presents the $ServTicket$ from M4 to B along with a new $authenticator$. In practice, other information may be sent to B too.

ii) (optional) B replies with a manipulated $\mathcal{T}_4$, thus authenticating the service.

Multiple Realms/Kerberi

- A *realm* is defined by a Kerberos server (KAS + TGS). It stores the user passwords and application server keys for its realm.
- Large networks may be divided into administrative realms.
- Inter-realm protocol:
  1. Servers register symmetric keys with each other.
  2. For A to access B in another realm, the TGS in A's realm receives the request and grants a ticket to access the TGS in B's realm. A then requests access at TGS in B's realm, who issues a $ServTicket$ for access to B.
- Simple protocol extension (two additional steps); yet for n realms key distribution is $O(n^2)$ (in Kerberos v4)

Limitations

- M1: encryption is not needed, but attacker can flood the KAS (DOS opportunity)
- M2: double encryption is redundant, eliminated in Kerberos v5
- Relies on (roughly) synchronized and uncompromised clocks.
  If a host is compromised, then its clock can be compromised and replay is trivial using old tickets and authenticators.

9.2. OAuth 2.0

Terminology

- Resource Owner RO (sometimes called User) – you
- Resource Server RS – e.g. mail.google.com
- Authorisation Server AS – e.g. accounts.google.com
- Client – e.g. Thunderbird

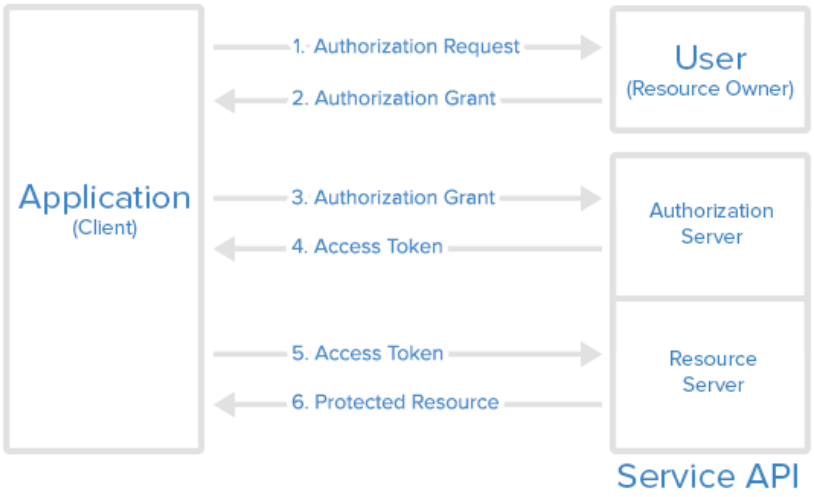

Figure 28: OAuth high level protocol flow

All messages are sent via TLS channels!

Client registration Clients must be registered ahead of time with each AS they want to service.

Client provides AS with:

- Application name
- Application website
- Redirect/Callback URL $\longrightarrow$ only here AS will send access tokens to

Client receives from AS:

- Client ID
- Client secret

Authorisation grant types The client exchanges the authorisation grant (received from the RO) for an access token at the AS.

- *Authorisation code*: for server-side apps
- *Implicit*: for mobile and webapps
- others exist but should not be used anymore

Authorisation code grant flow Flow for a server application to obtain an authorisation code grant. It can then "trade" this for an access token to access the protected ressource (omitted here).

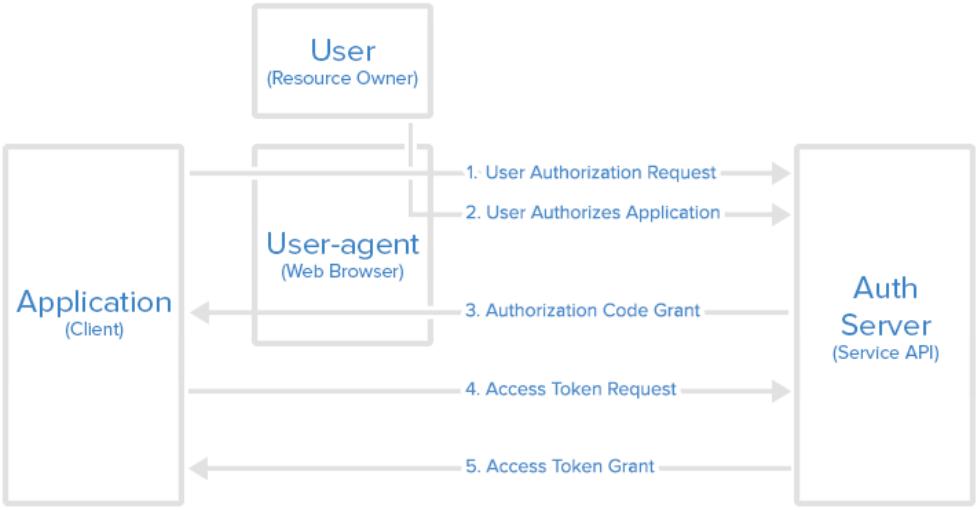

Figure 29: Authorisation code grant flow

Steps in Figure 29:

(1) $C \to AS : clientId$, 'give access to sth'

(2) not specified in OAuth 2.0

(3) $AS \to C : randomauthcode$

(4) $C \to AS : randomauthcode, clientsecret$

(5) $AS \to C : token$ (see below; goes directly to C's callback URL)

ad (2):
User receives a popup **in the AS context** that requests access for the client, valid at RS, with specific permissions and for limited time. User-to-AS authentication is out of scope and handled by the AS. After user grants the access, the authorisation code *randomauthcode* is sent to the application.

ad (4):
Application contacts AS directly with *randomauthcode*. AS requires this to be bound to a Client ID, through proof of *clientsecret*.

ad (5):
AS responds – in a separate TLS session – to the known Callback URL. This enforces that the sender is indeed Client, by ownership of that URL. AS issues an Access Token, which is possibly audience-restricted to the Client (see below).

Implicit grant flow see Figure 30. Covered in detail in the exercise

Access token

- Many different versions/formats specified
- Bearer tokens are possible (c.f. a Swiss franc coin)
- Mostly signed JSON Web Tokens (JWT[22]) used nowadays

[22]Pronounced "jot"

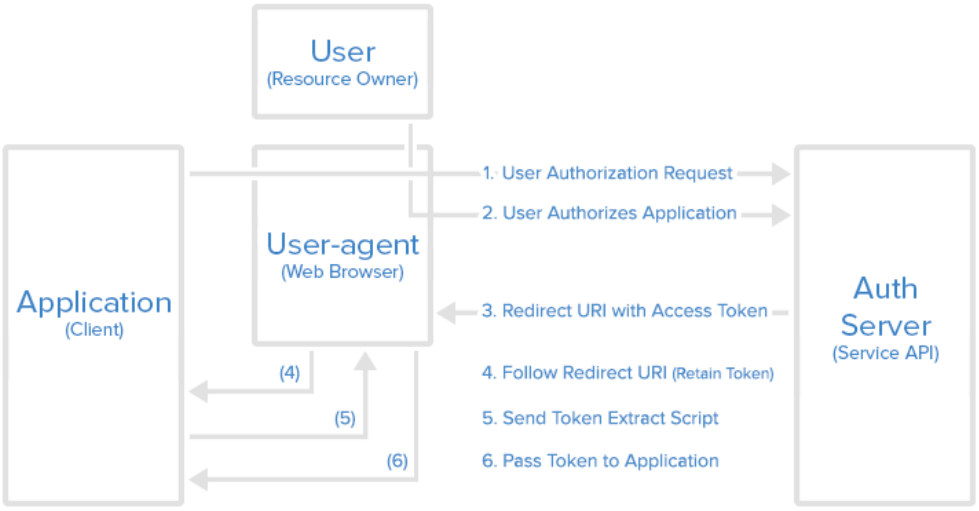

Figure 30: Implicit Flow

- Includes "claims" which limit audience, duration, access level
- Signature prevents fake tokens
- Audience limitation prevents misuse of accidentally leaked tokens, as it requires additional Client verification

In practise OAuth 2.0 has widespread adoption for delegated authorisation. Can it also be used for authentication (of the user against the client)?

- Yes, BUT it should not (directly)!
- ↯ Access tokens are not proofs of authentication (they're opaque):
  They are bound to a Client (who is the presenter of the token). The token audience is the RS, not the Client!
- ↯ Accessing protected resource does not mean the user is present (tokens are long-lived!)
- Properly building authentication on top is possible: Facebook Connect, Sign in with Twitter, OpenID Connect

9.2.1. OpenID Connect OIDC

The AS acts as the *Identity Provider IdP* and authenticates a user for the client (herein called the *Relying Party RP*). For the protocol flow, see Figure 31[23].

- Difference to OAuth 2.0: sends `scope=openid` with initial auth request, later gets an `id_token` back alongside the access token
- ID tokens are signed JWT with the following fields:
  - *sub*: subject (user identifier)
  - *aud*: audience (RP, e.g. ETH Gitlab)
  - *iss*: issuer (IdP, e.g. Google)

[23]Mapping this chart to the OAuth 2.0 flow from Figure 28: (2),(3) ↦ 1. and (6),(7) ↦ 4.

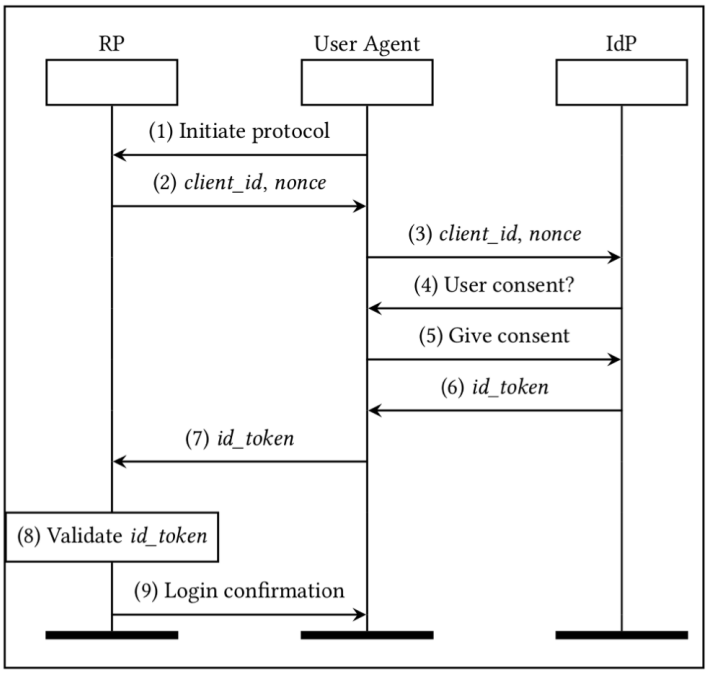

Figure 31: OpenID Connect flow

  - Token lifetime (short!)
- Supports dynamic IdP server discovery and client (not user!) registration on-the-fly
- Drawback: the IdP learns every log-in the user performs, as well as all sites visited
- ID token header, JWT object, and signature are encoded into a base64 URL-safe string. This allows easy passing around as an URL parameter.

It is easy to confuse the following: OpenID does authentication. OAuth 2.0 does authorisation. OpenID Connect extends OAuth 2.0 and does both authentication and authorisation at the same time.

9.2.2. Comparison: Kerberos versus OAuth 2.0

- Kerberos intended for unified corporate networks, not for internet and cross realm use
  $\Longrightarrow$ Registering new applications is difficult (because of key sharing TGS $\leftrightarrow$ service)
- Containerised apps are not possible with Kerberos since it needs secrets[24][25]
- Kerberos covers: authentication, authorization, audit[26]
- OAuth 2.0 covers only authorization, but extensible to authentication with OIDC
- OAuth 2.0 is flexible – applications register at services

[24] thgoebel: I disagree, see `docker secret create` or `kubectl create secret`

[25] eisman: Disagree: OAuth also requires a client ID–client secret pair. Regardless of that containerized applications are well possible with Kerberos

[26] Audit is not covered by the protocol but can be done manually.

9.3. SSL/TLS

Goals Secrecy, integrity and authentication (optionally mutual)

SSL Handshake Subprotocol for initiating the connection: establishing keys and verifying identities.

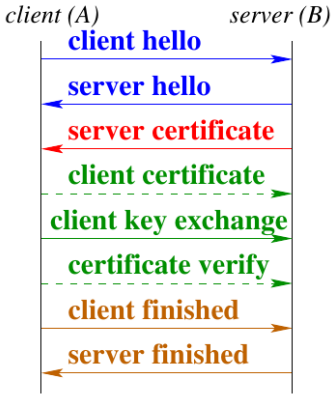

Figure 32: SSL handshake

1. client hello $A \to B:\quad A, Na, Sid, Pa$
   server hello $A \to B:\quad Nb, Sid, Pb$

   A identifies itself, sends its nonce Na, a session id Sid, and its preferences Pa for key exchange, encryption, compression. From that the server chooses the preferences Pb to be used.[27]

2. server certificate $B \to A:\quad certificate(B, K_B)$

   Server identifies itself through a certificate (in X.509 format) containing – amongst others – its name and public key.

3. client certificate (opt) $A \to B:\quad certificate(A, K_A)$
   client key exchange $A \to B:\quad \{PMS\}_{K_B}$
   client verify (opt) $A \to B:\quad \{hash(\dots)\}_{K_A^{-1}}$

   Client sends the presudo-random *Pre-Master Secret* PMS which is used to derive a master secret M.

   Possibly – but rarely done in practice – the client also sends its certificate as well as a signed hash of all the previously exchanged messages.

4. client finish $A \to B:\quad \{Finished_A\}_{clientK}$
   server finish $B \to A:\quad \{Finished_B\}_{serverK}$

   $Finished$ is a hash of all previously exchanged messages. It authenticates the server and guarantees integrity of previous messages (e.g. preventing downgrade attacks).

   $clientK$ and $serverK$ are symmetric keys derived from Na, Nb, M for encryption. Two additional keys for MACs and two stream cipher IVs are also derived. These will be used througout the following session.

SSL Record Subprotocol for sending data: detailing compression, authentication and encryption.

[27] Notabene: Pa and Pb are not secured at this stage but only authenticated in the $finish$-stage.

SSL session Client–server association, together with some state (encryption method, encryption keys, MAC secrets, ...)

SSL connection Secure stream within a session.

TLS v1.3 Removes a lot of unused and unsafe features. Combines client hello and key exchange (aka. key share), allowing a 1-RTT handshake. If agents have a common pre-shared key PSK then 0-RTT is possible at the expense of perfect-forward secrecy PFS.

Problems In practice, TLS only provides *server-side authentication.* This enables MITM attacks and phishing: evildoers can spoof a real site (through social enginnering, DNS poisoning, ...), sitting in the middle and relaying traffic on both ends over TLS.

Solutions (each with their own drawbacks) include:

- *Multi-channel authentication*: e.g. send TAN via SMS. Fails for user authentication – the attacker can just pass through the entered TAN! But okay for transaction authentication (e.g. include amount and destination of bank transfer in SMS).
- *Session-aware user authentication*: In $finish$ stage when checking the hash of previous messages, check that the session where the user send the credentials is the same session in which the server receives them.
- *Client certificates*: As originally designed (see handshake stage 3). However, has problems with usability and PKI rollout.

9.4. IPSec

Goals End-to-end security between client–server or gateway–gateway. Message authentication + encryption. Protect packet headers. Traffic filtering.
$\Longrightarrow$ too many features, high complexity

Security association Stores metadata required for communication: protocol format, algorithms, keys, etc. Either set up manually or negotiated through invocation of IKE (see Section 9.4.1).

Protocol modes

A. *Transport*: between client and destination server. Payload encrypted/authenticated and IPSec header inserted.

B. *Tunnel*: between two gateways or between endpoint and proxying gateway. Payload protocol encapsulated in second delivery protocol.

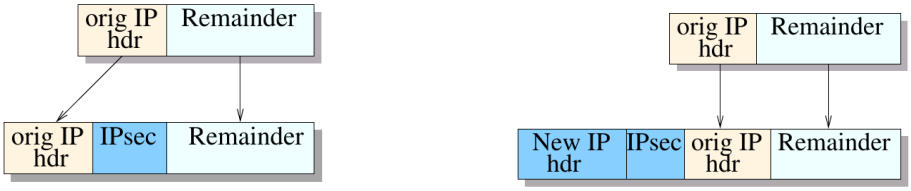

Figure 33: IPSec transport (left) and tunnel mode (right)

Headers

- *Authentication Header AH*: protects packet integrity and authenticity, NOT confidentiality
- *Encapsulating Security Payload ESP*: protects confidentiality (and optionally integrity)

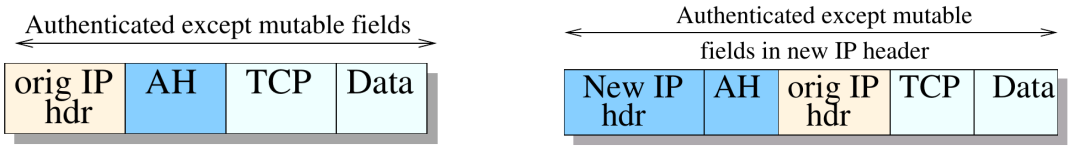

Figure 34: AH header in transport (left) and tunnel mode (right)

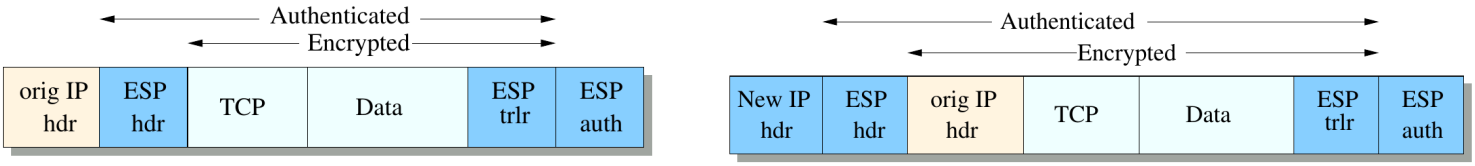

Figure 35: ESP header in transport (left) and tunnel mode (right)

Security Policy Database Database with rules (*Condition*, *Action*) on how to handle packets: allow or block? AH or ESP? Transport or Tunnel? Known security association? Manually configured by administrator.

9.4.1. Internet Key Exchange Protocol IKE

Note that IKE is an application layer protocol while IPSec belongs to the transport layer!

Purpose Establish shared keying material, e.g. security associations for IPSec. Like IPSec very complex with a lot of options and features. Based on Diffie-Hellman key exchange (see Section 5.1).

Solving practical DH problems

- Trivial MITM attack: Eve negotiates separate keys with Alice and Bob, to later de- and re-encrypt their traffic.

  Solution: TODO slide 40

- DOS attack: flood server with randomly generated h_1s (Xs) – without ever computing g^x – from many spoofed IP addresses. Server incurs load of exponentiation and storing state.

  Solution (partially): Either demand response from claimed address, or require initiator to perform some work.
  The latter can be done using "cookies": initiator I and responder R first exchange cookies C_1, C_2. Ideally, cookies are stateless: $C = hash(IP,\ localSecret)$. Now the attacker must be at the IP addresses (to receive C_2) and complete the cookie exchange.

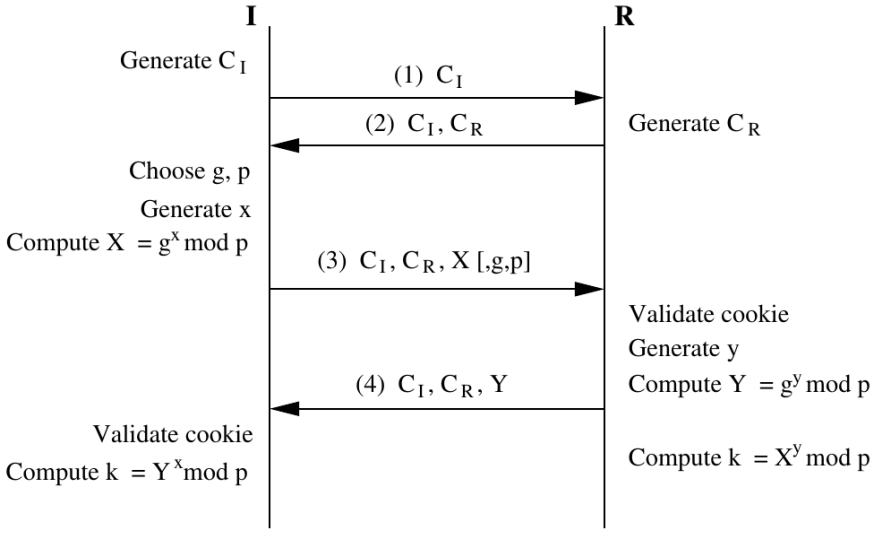

Figure 36: DH key exchange with cookies

Perfect Forward Secrecy Property in which an attacker who records an encrypted conversation cannot later decrypt it even if she has since compromised each side's long-term keys.

IKE achieves this by having pre-shared keys (aka. master keys) and then performing a DH key exchange in each protocol run.

IKE phases

1) Two parties negotiate SA. Outlined below. Has different modes:
   - Main mode: has identity protection and flexibility wrt. parameter choices
   - Aggressive mode: without identity protection

   Each mode has in turn four variants, based on the authentication method.
2) Use SA from phase 1 to create "child SAs" to use in further communication. Not explained here.

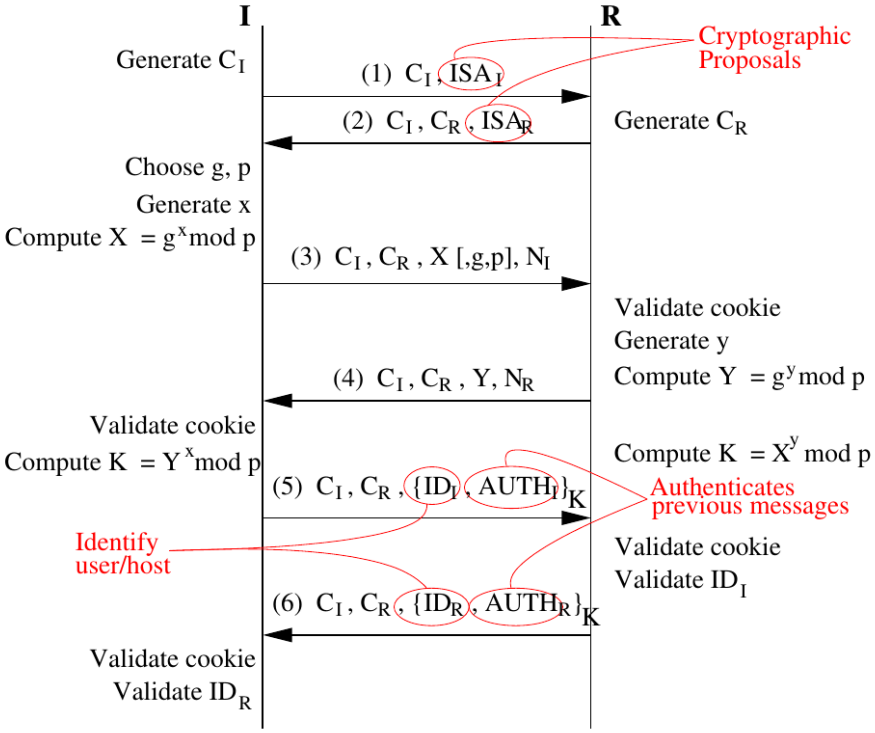

Figure 37: IKE phase 1

ad (1):
ISA_I includes a series of *proposals* from I, listing supported alogrithms

ad (2):
ISA_R is the proposals accepted by R and the algorithms chosen

ad (3), (4):
Contains public keys and nonces of I, R

ad (5), (6):
Encrypted using algorithm negotiated in (1), (2) and key K derived from DH exchange in (3), (4). ID_I, ID_R are some identifiers (e.g. IP address, fully-qualified domain name, certificate, email address). $AUTH_I, AUTH_R$ are authentication data for previous messages (e.g. signature, MAC).

10. Public Key Infrastructure PKI

Key management

- Bind key to an identity/principal (and purpose)
- Distribute keys
- Generate, maintain, revoke keys

For symmetric keys, the standard solution is a trusted third party acting as a key server. In a star-shaped setup it shares the keys with each principal. However, for public keys the range of options is much wider.

10.1. Certificates

Certificate Data structure that binds an identity to a key. Consists of the identity, its key, some parameter values and all of the above hashed and signed by some trusted authority C. In order to validate the certificate, one must know C's public key K_{CA} and trust C to correctly check the binding before creating a certificate.

X.509 certificate format Commonly used format for certificates in TLS, IPSec, S/MIME, etc, though originally designed for the X.500 family of ITU standards.

See Figure 38. Notes for that diagram:

- $(serialnumber, issuername)$ must be unique
- *Subject name* is the X.500 public directory name of the client
- *Signature* is the signed hash of all other fields
- *Issuer/subject unique identifier* are optional, in case the X.500 name is ambiguous

Domain-validated certificate (DV) CA validates domain ownership, e.g. by email or by verifying control over the domain's DNS entries (c.f. Let's Encrypt's `certbot`). What you normally have.

Extended-validation certificate (EV) CA contacts the organisation and performs more checks (legal rights to name, etc). No security benefit, otherwise only marketing aimed at customer trust (because you spent more money). Results in green organisation name next to the lock icon in browsers.

10.2. Trust models

Trust in an entity: your believe that an entity behaves as expected.

Trustworthiness of an entity: whether the entity actually behaves as expected.

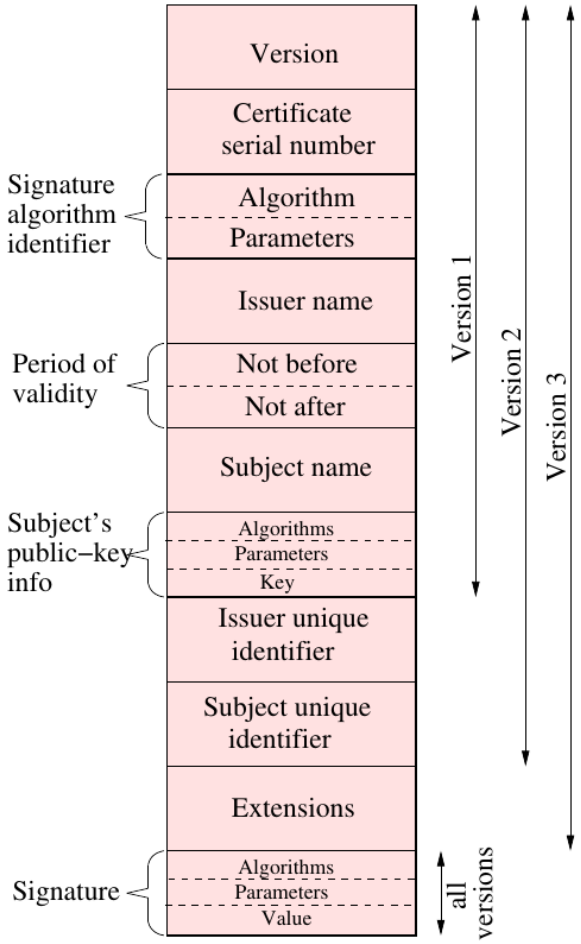

Figure 38: X.509 certificate structure

Certificate chain

To verify a certificate you need a public key.

Base case: you have it already.
Recursive case: you have other certificates that certify the public key that you need for verification.

This yields **certification chains**. Trust models dictate how chains are constructed and how users establish certificate validity.

Thus there is *direct* and *indirect* trust: trust that X correctly issues certificates, and trust that Y (recommended by X) is trustworthy to issue certificates.

Root of trust Start of the certificate chain, i.e. base CA that a user must trust and know the public key of.

Single CA model Single CA for the entire world.

- ⊕ Short/direct certificate chain
- ⊕ No need for trust recommendations

⊖ No single organisation trusted by everyone in the world

⊖ Inconvenient, insecure expensive to obtain certificate from distant CA

⊖ Difficult to change CA's keys

⊖ Monopoly

Single CA + RA model Single CA and multiple registration authorities (RA). The RA are trusted by the CA to perform the name–key binding verification on its behalf. RA signs request and sends it to CA for certification.

⊕ Users can interact with local RA

⊕ Need to know only a single public key (the CA's)

⊕ Easy revocation of compromised RA keys: notify CA

⊖ Other disadvantages of Single CA remain

⊖ CA must trust all RAs.

Oligarchy of CAs model Multiple (but few-ish) CAs. Users keep set of CA public keys.

⊕ Users can interact with local CA

⊕ Healthy competition bewteen CAs

⊖ Less secure: more CAs = more opportunities to compromise some CA

⊖ *Certificate store* of clients can be manipulated: public machines, Lenovo Superfish, eDellRoot

Variants (that can be combined!) include:

- *Cross-certification*: CAs mutually certify each others' keys. Thus a user need know all CA's keys anymore.
- *Namespaces*: CAs is only trusted to certify certain bindings, e.g. ETH CA for `*.ethz.ch` domains. Can be hierarchical.

Configured + delegated CAs model CAs configured in users' certificate store can authorise other CAs to act as delegated CAs. When, in turn, those authorise other delegated CAs, longer certificate chains are built. This is what is done in practice today: browsers and OSes have a certificate store.

⊕ Users can interact with local CA

⊕ Users need only be configured with few CAs keys'

⊖ As before: more CAs = more opportunities to compromise some CA

⊖ Based on strong assumption that trust is transitive

Web of Trust model Instead of CAs, people verify and recommend each other. You can have two trust levels for a key: either you trust a key for communication only, or – stronger – you trust a key to sign other keys (i.e. you believe its owner diligently checks identities prior to signing a key).

This is used by PGP. There a *keyring* file stores all known information. Each keyring entries stores: a) whether I believe the key to be authentic, and b) the level of trust I place in its owner to sign/certify other keys: complete, marginal, no trust.

You trust keys if they are signed by at least a single party that you trust completey, or at least two parties that you trust marginally.

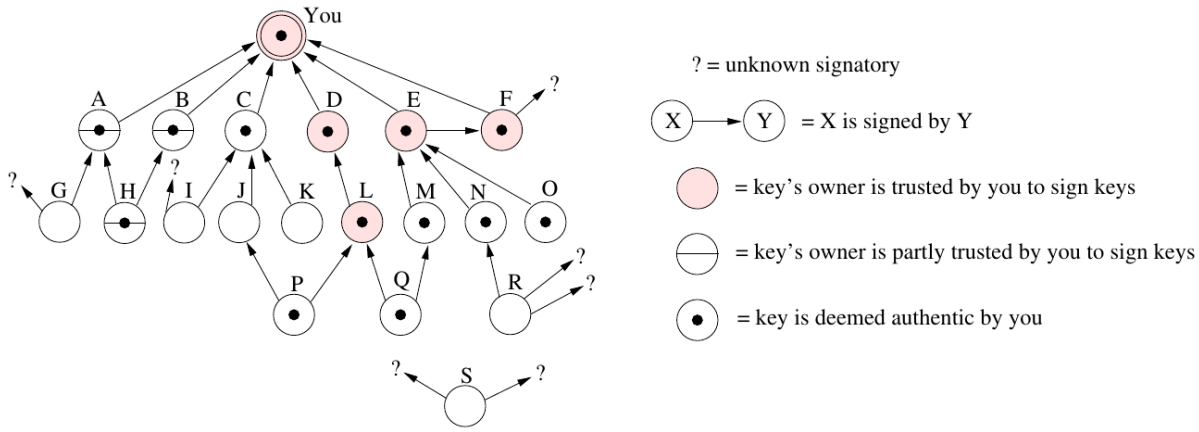

Figure 39: Web of Trust example

⊕ Avoids CAs and trust in organisations

⊖ Scales poorly beyond small communities. Hard to find certificate chains when users only sign ±10 others.

⊖ Trust can be misplaced

⊖ Usability: users need to understand the concepts and actively engage in signing keys and verifying signatures

10.3. Preventing fraudulent certificates

Fraudulent certificates CAs certifying wrong keys, either by error or intent. Happens regularly, resulting in (possible as well as actual) MITM attacks on TLS.

Classes of approaches There are three main classes of approaches:

- Client-centric
- CA-centric
- Domain-centric

Perspectives Globally distributed *notary servers* contact known TLS servers once a day, storing the observed certificate. Users configure and check a set of trusted notary servers.

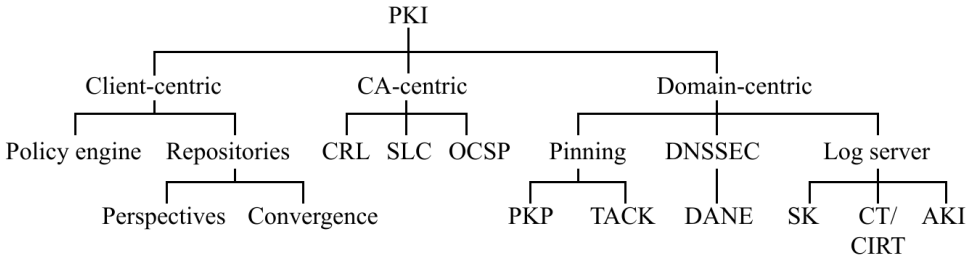

Figure 40: Classification of PKI improvements

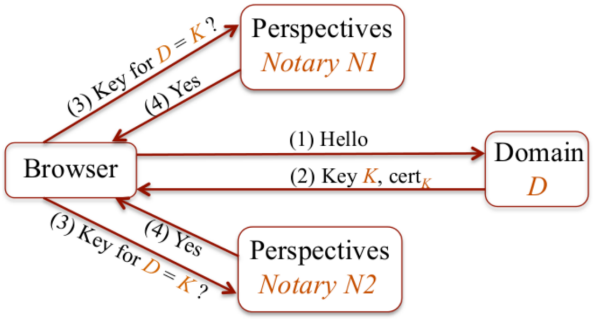

Figure 41: Perspectives

Convergence Solves Perspectives' privacy issue using an onion-routing based lookup: user send an encrypted request to the first notary that forwards it to another notary that then performs the key check.

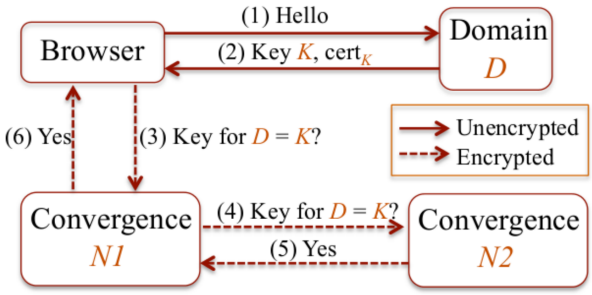

Figure 42: Convergence

Key pinning User "pins" a certificate to a domain (stores the association) and exclusively accepts those. Either preinstalled/hardcoded, or trust-on-first use. Has numerous disadvantages leading to lockout of users, thus abandoned by Google Chrome in 2018 in favour of CT. Still common practice in controlled environments such as mobile apps.

DNS-Based Authentication of Named Entities DANE Domain owners publish the server's certificate fingerprint in a DNS record (secured with DNSSEC). Clients can check the fingerprint during the TLS handshake.

Domains may also specify the CAs that should create certificates for in a `CAA` record.[28]

[28]Though this does not prevent a malicious CA from doing it anyway.

Certificate Transparency CT aims at making attacks publicly visible and thus deterring them. CT requires a certificate to be in a public log before a browser will accept and use it. It allows removal of malicious certificates (though domain owners must actively check the logs) or even of malicious CAs.

See protocol flow in Figure 43. Logs are implemented as append-only Merkle Hash Trees (MHT). Together with the certificate, domain owners receives a *Signed Certificate Timestamp SCT*: a cryptographic promise by the log server to add the cert to the logs (done periodically). Browsers check that the SCT is correctly signed by the log server for the right certificate and that the timestamp is not in the future. For the Monitor and Auditor roles supervising the log servers, see RFC 6962.

Disadvantages: requires active log-watching, does not prevent attacks, does not support certificate revocation.

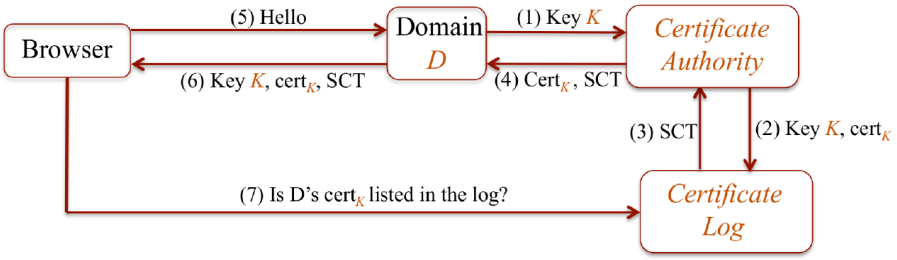

Figure 43: Certificate Transparency

Attack-Resilient Public Key Infrastructure ARPKI Similarly, a log-based approach, with clients requiring certificates to be in the log. Additionally, requires at least n CAs to sign a certificate before it is considered valid (default: $n = 2$). Stores domain's policy regarding trusted CAs, revocation parameters, etc.[29]

10.4. Other issues

Naming and identity Unique naming is difficult. Names, attributes, identities, roles change over time and often depend on the context.

Revocation CAs maintain and sign a publicly accessible *Certificate Revocation List CRL*, listing all revoked but un-expired certificates issued by that CA.

The *Online Certificate Status Protocol OCSP* can be used to inquire about a certificates status: good/revoked/unknown. However in practise, that adds another roundtrip, and no browser even treats OCSP errors as fatal.
The roundtrip issue is addressed by *OCSP stapling*: the CA signs the OCSP information and attaches it to the certificate. The server sends both to the client, improving latency and privacy (no client contact with OCSP server).

Key recovery The CA may store the keys for different reasons:

- *Key backup*: recovery of lost keys
- *Key escrow*: storing keys for government access

[29] thgoebel: Unclear to me where the policy is stored: in the certificates, the log servers, or the "ARPKI infrastructure"?

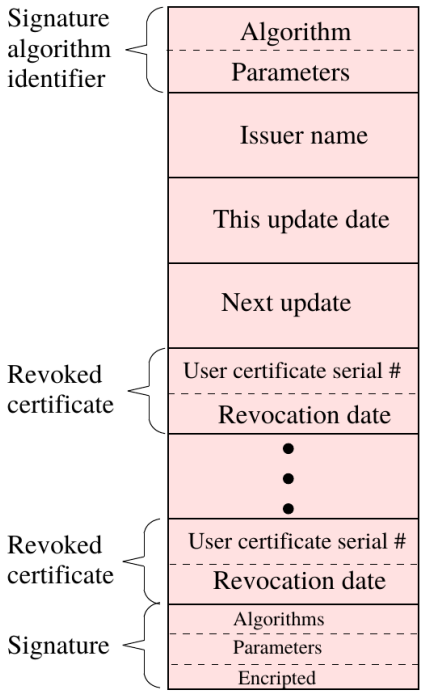

Figure 44: X.509 certificate revocation list format

To reduce the trust needed, split the key into n shares stored by different parties that need to collude in order to recover the key.

Content Delivery Networks CDNs serve content on behalf of domain owners using a network of geographically distributed servers.

Subject Alternative Names SANs allow a certificate to be valid for many different domains. Domain owner Alice authorises the CDN to get a certificate for her domain with the CDN's key. Disadvantage that a CDN certificate is much more precious now that it is valid for many different domains.

11. Access Control

11.1. AAA

Identification The process of associating an identity with a subject.

Authentication The process of verifying the validity of something claimed by a subject (typically their identity).

Mechanisms for identity authentication:

- Something you know: e.g. PIN, password
- Something you have: e.g. smartcard, Yubikey
- Something you are: biometric characteristics, e.g. fingerprint, face, iris

Authorisation The rights or permissions that are granted to subjects to access system resources.

Can be specified using mathematical structures, rules, programs, etc.

Simplest case: A relation (state) $AC \subseteq S \times O \times R$, specifying rights ($R$) that subjects ($S$) have on objects ($O$). Administrative actions cause state transitions.

Often more complex in practice. Can be dependent on subject/object/environment attributes and history, and even entail obligations for the future (see rest of this section).

The above concepts are independent. E.g.:

Authentication without identification: using a stolen password

Authorisation without Authentication: having a physical key to a building

Access Control Protection of system resources against unauthorized access.

Namely, the process and controls for regulating the use of system resources by means of a security policy whereby access is restricted to those entities authorized by the policy.

Deployed at many hardware and software levels. Mechanisms include:

- Memory management hardware
- Operating system, file system, and related services
- Middleware
- Application
- Firewall
- Physical protection

Design principles:

- *Complete mediation*: access to every object must be checked.
- *Minimally trusted computing base*: mechanism should be as simple and reliable as possible.

Challenges: implementation bugs, layer-below + side-channel attacks, faulting user

11.2. Basic Concepts

Security policy Defines what is allowed, i.e. which system executions are (in)acceptable. Analogous to a set of laws and defined in terms of high-level rules or requirements.

TLDR: Defines a set of authorised states or traces/histories.

Security model Provides a formal representation of a class of systems and their behavior, highlighting their security features at some chosen level of abstraction.

System state The collection of current values of all memory locations, storage, registers, and other components.

Protection state System part relevant for security. E.g. part of system state determining who is reading/writing files.

Transition system Abstraction capturing a system's dynamics through changes in the system state.

Authorized states Subset Q of protection states P in which the system is authorized to reside in. That is: authorized states: $Q \subseteq P$, unauthorized states: $P \setminus Q$.

- A security policy characterizes Q. Hence a policy partitions the system's states into authorized states and unauthorized states.
- A security mechanism must prevent a system from entering $P \setminus Q$.
- A secure system is one that starts in an authorized state and can never enter an unauthorized state.

11.3. Access Control Matrix Model

Access Control Matrix Model Describes a protection system by describing the privileges of subjects on objects.

- Subjects S: users, processes, agents, groups, ...
- Objects O: data, memory banks, other processes, ...
- Privileges P (aka permissions/rights): read, write, modify, ...

Protection state in ACM ... is a triple (S, O, M) – made up of a set of current subjects S, a set of current objects O and a matrix M defining the privileges for each $(s, o) \in S \times O$. That is, it is a relation $S \times O \times P$ or equivalently a function $S \times O \to \mathcal{P}(P)$.

Commands State transitions model administrative actions and are formalised by a set of commands. Commands are expressed in terms of 6 primitive operations:

1. **enter** p **into** $M(s, o)$
2. **delete** p **from** $M(s, o)$
3. **create subject** s
4. **destroy subject** s
5. **create object** o
6. **destroy object** o

Transition system semantics

- $(S, O, M) \vdash_c (S', O', M')$ denotes a transition associated with the command c
- A starting state $st_0 = (S_0, O_0, M_0)$ and a set of commands C determines a state-transition system.
- A model describes a set of system traces, namely those traces $st_0, st_1, \ldots$ where $st_i \vdash_{c_i} st_{i+1}$, for $c_i \in C$.

Thus we get a class of access control system models parameterised by a set of privileges, a set of commands, an initial state and a universe of subjects and objects.

A system is a model instance, describing what subejcts can do (*mechanisms*) not necessarily what they are authorised to do (*policy*).

Correctness Let S be the set of all possible system states, P the set of authorised states (given a policy) and R the set of reachable states.

Then a system is *secure* if $R \subseteq P$. It is *precise* (not overly protective) if $R = P$.

Often, authorisation depends on the past, and policies are defined on traces. Then a system is secure if every possible trace is authorised. However, this is hard to represent in the matrix model (see security automata later).

Leakage safety A state st is leakage safe for a privilege p if there is no sequence of commands that – when started in st – will write p into a matrix cell that does no yet contain p.

More formally, there exists no st' such that $st \vdash^* st'$ and st' contains any new occurrence of p.

Undecidability of leakage safety Theorem by Harrison-Ruzzo-Ullman, 1976. Proof proceeds via reduction to halting problem. Note that this theorem is only for the matrix model class of systems.

Given arbitrary $\vdash, st, p$, it is undecidable whether the state st is leakage safe for p.

Mechanisms Different implementation approaches have different tradeoffs, pros, and cons. Exercise: list them (hint: storage space, complexity to update (add/revoke) and query, scalability).

One advantage of ACL is **discretionary access control**: owners have the authority to delegate rights on objects they own to other subjects.

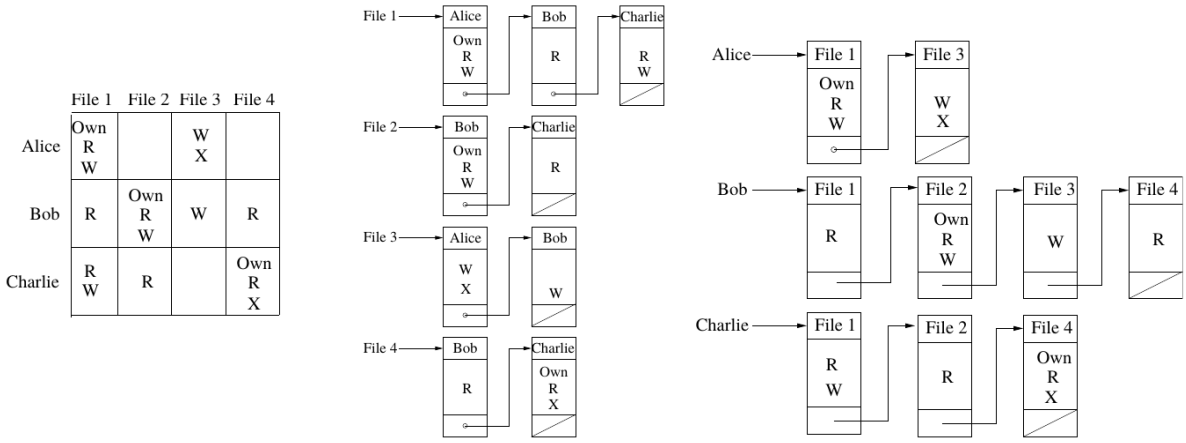

Figure 45: Access Control matrix, access control list, capabilities list

11.4. Role-based Access Control RBAC

Motivation & Idea AC matrices scale poorly and are difficult to maintain. Luckily in reality, users have certain **roles** within an organisation. Instead of assigning privileges to users directly, we can assign users to roles, and give certain roles certain privileges.

Formal definition RBAC is defined by a set of of roles and the two relations $UA \subseteq Users \times Roles$ and $PA \subseteq Roles \times Privileges$, such that $AC \coloneqq PA \circ UA$. In other words:

$$AC \coloneqq \{(u,p) \in Users \times Privileges \mid \exists r \in Roles : (u,r) \in UA \wedge (r,p) \in PA\}$$

Role hierarchies Formally: $AC \coloneqq PA \circ \geq \circ UA$. Semantically, larger roles inherit privileges from smaller roles (e.g. a superuser has all ordinary user rights).

Pros and cons

- ⊕ Within a real organisation, roles are stable, whereas users come and go
- ⊕ Supports principles of *least privilege* and *separation of duty*[30]
- ⊖ No fine-grained access control (e.g. teacher should only grade her own class)
- ⊖ No access control based on system state (time of day, available funds, ...)
- ⊖ No access control based on traces/histories
- ⊖ No support for delegation
- ⊖ Cannot specify **obligations** arising from granted access

But RBAC may be combined with programmatic access control, checking system state at runtime.

[30] Achieved by requiring two mutually exclusive roles to complete a task.

11.5. Discretionary Access Control DAC

Idea Owner can change an object's permissions at her **discretion**. E.g. Unix file permissions.

Pros and cons

- ⊕ Allows direct delegation of rights
- ⊕ Allows transfer of ownership
- ⊖ Open to mistakes → all users must understand and respect the mechanisms and security policy
- ⊖ No central control
- ⊖ Trojan horse problem (malware happens...)

11.6. Mandatory Access Control MAC

Mandatory System owns resources and controls access. Thus power is shifted from users (c.f. DAC) to the system operator.

Labels/Classifications Basis for access control decisions. Associated with subjects and objects. A label is a pair of:

1. *Sensitivity/clearance*: e.g. Top Secret ≥ Secret ≥ Confidential ≥ Unclassified[31]
2. *Compartment/category*: domain for "need-to-know" policy, e.g. Europe, Middle-East, Satellite

Subjects can read objects up to and including their clearance level.

Fun with maths Clearances and categories form lattices: subjects are only authorised to read objects if their label dominates the object's label.

Recall that a *partially ordered set (poset)* is $(S, \leq)$ where where S is a set and relation $\leq$ is reflexive, transitive and anti-symmetric.[32] Furthermore, recall that a *lattice* is a poset where each $a, b \in S$ has a least upper bound as well as a greatest lower bound.[33]

The *dominance* relation is defined component-wise on linearly-ordered clearances and subset-ordered category sets:

$$(r_1, c_1) \leq (r_2, c_2) :\Leftrightarrow r_1 \leq r_2 \wedge c_1 \subseteq c_2$$

Thus lattices are well-suited for need-to-know policies, since we can uniquely answer questions like "What is the minimal label a subject must have to read two objects with different labels?" or "What is the maximal label of an object such that two given subjects can read it?"

[31] Note how this is obviously historically/militarily inspired.
[32] Exercise: Write down the definitions for those three.
[33] Exercise: Again, write down the definitions.

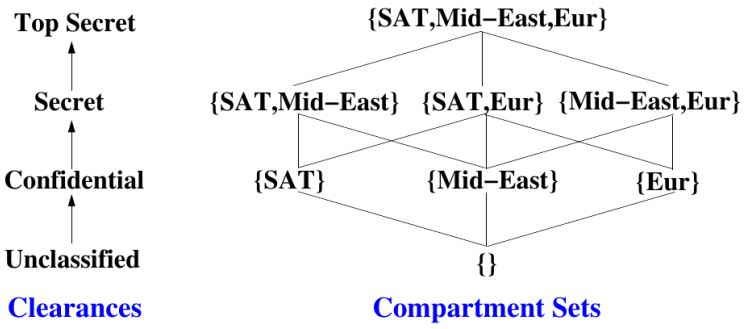

Figure 46: Example relations for clearances and categories

Bell-LaPadula BLP model Models security policies for *confidentiality*. The main challenge is to prevent *information flow*.

Access decisions satisfy the following properties:

- *Read-down (aka Simple Security Property):* a subject with label x_s can only read from an object with label x_o if x_s dominates x_o.
- *Write-up (aka *-Property):* a subject with label x_s can only write to an object with label x_o if x_o dominates x_s.

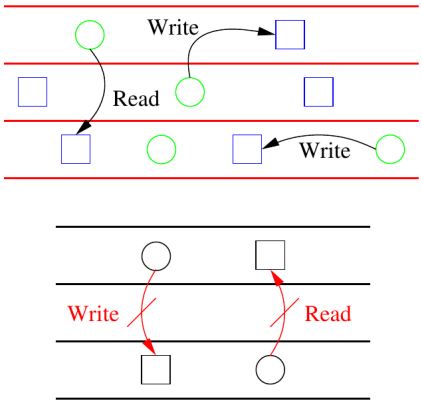

Figure 47: (Dis) allowed operations in Bell-LaPadula

Pros and cons:

⊕ Disallowing write-down ensures confidentiality by preventing reclassification to lower clearances

⊕ No information leakage possible (read-down + write-up, plus labels cannot be changed by users/malware due to MAC)

⊖ Prevents "legitimate" downwards communication (e.g. General → Lieutenant)

Possible solutions:

- *High-water mark* (BLP's solution): Subject s has 2 labels: $curlevel(s) \leq maxlevel(s)$. We set $curlevel(s)$ to the clearance that dominates all objects that s has read so far. s can write to $curlevel(s)$ and above.
  ↯ High-water mark keeps rising
- Temporarily downgrade subject's clearance

- Specify a set of trusted subjects that may violate the *-Property under some condition, e.g. when involving $n > 1$ trusted subjects (Four-eyes principle)

Examples for BLP in practise include Sun Trusted Solaris, Trusted HP-Unix and NSA's SELinux.

Integrity models Programs process data from multiple sources – what is the *integrity* of the result? (Analogy to water: drinking water can be contaminated with a single drop).

Low-water mark model: Object's integrity is the lowest integrity level of all objects interacted with.
↯ Too simplistic – everything gets contaminated.

Biba Integrity Model: Dual to BLP: information may only flow from high-integrity to low-integrity regions. That is, only *write-down* and *read-up* are allowed.
Example: Windows's Mandatory Integrity Control: untrusted software (Internet Explorer) runs at low integrity level, and since user files have medium integrity it cannot overwrite them.

Chinese Wall Model Avoids conflict-of-interest situations by preventing crossing to the other side of the wall.

Commercially inspired: A subject (UBS employee) may only access data iff she has not previously accessed data of another company in the same conflict-of-interest class (e.g ZKB), i.e. iff she is on the right side of the wall.

Note how access rights change over time, with each access.

Interface models Previous AC models only restrict read/write – but information can leak in other ways: e.g. CPU usage, type of errors, file lock/unlocked, etc. Rather than restricting operations, interface models specify restrictions on a systems input/output. One possible property to restrict is:

Non-interference
Intuitively: A's behaviour must not change B's view of the system.

Definition: Security property, under which a group of subjects using a set of commands is *non-interfering* with another group of subjects if whatever the first group does has no effect on what the second group can observe.

Subjects can be users, processes, program variables, etc. Commands can be actions, variable updates, etc.

Formally: Associate a security levels with subjects s and with inputs to the functions f_s executed by s. If s has level l then the result of f_s must only depend on inputs dominated by l.

Anti-example, not satisfying non-interference: $f_s(..., x, ...) =$ if x then 1 else 0 and assume that x has level $high$ and s has level low.

Formally #2 (Goguen & Meseguer): Let U be the set of users. Let $out(u, I)$ be the system's (deterministic) output function whose value is the output to user u generated by the input history $I = i_1, ..., i_n$. Let $purge(u, I)$ purge/delete from I all inputs i_j where the clearance of u' who input i_j dominates the clearance of u. The system is non-interfering if

$$out(u, I) = out(u, purge(u, I))$$

Research challenges: Formalising and proving non-interference for real systems. Often too strong a property – e.g. entering a password changes the output! Also what is observable at the interface - timing, power consumption, noise?

11.7. Monitor-based enforceability

Property A set is a *property* iff set membership is determined by each element alone, independent of all other elements of the set.

Notabene: Not all security policies are a property! E.g. non-interference is not, because whether a trace is non-interferent depends on other input traces (specifically, whether there is any other conflicting trace).

Execution Monitoring EM Basis for many enforcement mechanisms, such as reference monitors, firewalls, etc. Monitor system executions (traces) and allow/block them according to some security policy.

Formal setup: Let Ψ be the universe of all possible (in)finite traces. A security policy P is a predicate on sets of executions, i.e. it defines a subset of $\mathcal{P}(\Psi)$. A system S defines a set $\Sigma_S \subseteq \Psi$ of real executions. S satisfies P iff $\Sigma_S \in P$.

Which properties are EM-enforceable? Because EMs work by monitoring executions, any enforceable policy P must be specified such that

$$P(\Psi) \Leftrightarrow \forall \sigma \in \Psi . \hat{P}(\sigma)$$

where $\hat{P}$ is the formalised criteria used by the EM to decide whether a trace σ is allowed.

This gives rise to three requirements ($\tau \leq \sigma$ if τ is a finite prefix of σ):

1. P must be a property that is formalisable in terms of a computable predicate $\hat{P}$ on traces.
2. *Prefix closure:* if a trace is not in $\hat{P}$ then neither is any extension. Conversely if a trace is in $\hat{P}$, then so are all its prefixes. (Intuition: mechanism cannot decide based on future actions. System may terminate at any time.)

$$\forall \sigma \in \Psi . \hat{P}(\sigma) \rightarrow (\forall \tau \leq \sigma . \hat{P}(\tau))$$

3. *Finite refutabilty:* If a trace is not in $\hat{P}$, this must be decidable with a finite prefix.

$$\forall \sigma \in \Psi . \neg \hat{P}(\sigma) \rightarrow (\exists \tau \leq \sigma . \neg \hat{P}(\tau))$$

Safety property Nothing bad ever happens. Class of temporal properties. See FMFP for a proper recap.

If a set satisfies all three requirements above, then is is a safety property. Thus the properties that are EM-enforceable are exactly the safety properties. Similarly, if a set of traces for a security policy P is not a safety property, then no EM mechanism exists for P.

Invariants are a special case: they are predicates on states that must be satisfied in each execution step (and are violated in finite time).

Security automata model Possible implementation of an EM mechanism (e.g. by running the automaton in parallel with the system). Accepts safety properties. A security automaton $\langle Q, Q_0, I, \delta \rangle$ is defined by:

- a countable set Q of automaton states
- a non-empty set $Q_0 \subseteq Q$ of initial states
- a countable set I of input symbols
- a transition function $\delta : (Q \times I) \to \mathcal{P}(Q)$

A sequence $i_0, i_1, ...$ of input symbols is processed by a run $Q_0, Q_1, ...$ If Q_k is empty then i_k is rejected, otherwise accepted.

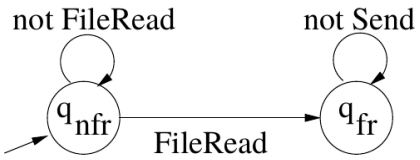

Figure 48: Oversimplified example of a security automaton

12. Privacy, Anonymity, Data Protection

What is collected when you surf?

- Every IP packet identifies sender, receiver, protocol.
- HTTP includes information on
  - IP address: 129.132.178.14
  - Host: zoe.inf.ethz.ch
  - Browser/OS: Mozilla/5.0 (X11; Linux i686; en-US; rv:1.1)
  - Referrer: google.com&q=David%20is%20very%20smart (i.e. URL of previous webpage, from which a link was followed)
- Cookies allow tracking within a domain.
- Information logged at routers and servers (even if TLS is used), e.g: your company/ISP gateway and servers you visit
- Payload information available at end points

Other data sources

- Financial: transfers (salary, expenses), investments, credit card purchases, ...
- Shopping: even with cash payments when using loyalty cards
- Telephone data: source, time, content (voice/data) if wiretapped. Also location profiles with mobile phones
- Medical data
- Online databases: news archives, social networks, ...

12.1. Definitions

Confidentiality No unauthorized access to information. Examples: mail, bank balance, e-votes.

- Requirements specify who can access what information. Example: mail must be unreadable in transit.
- Standard mechanisms exist. E.g. encrypt communication and control access through a server-side reference monitor.

Anonymity Identities of communicating agents are secret. Examples: E-complaint boxes, surfing, criminal transactions.

Data protection Collected data only used for limited, predefined purposes. Example: Customer email address used only for sending software updates.

Privacy

- According to Oxford English Dictionary: "A state in which one is not *observed* or *disturbed* by others."
- In the IT context this can be specialized as follows:
  - *Anonymity*: unobservability (or at least unlinkability) of our actions when they occur.
  - *Data Protection*: ensuring that our collected data is not distributed and used in undesired ways.

12.2. Anonymity

Anonymity within a group Assume your actions can be observed (e.g. sending/receiving). You are only anonymous within a group if your actions are indistinguishable from the actions of others in that group.

This group is called the **anonymity set**. The larger, the better.

You cannot have anonymity by yourself. Thus anonymity is strongest when the anonymizing service attracts many users.

e.g. using your home IP address (exposed/provably exposed) versus using the IP address of proxy (probable innocence for large enough subnet) versus using a mix network (absolute anonymity).

Anonymous communication Requires sender anonymity, receiver anonymity, or unlinkability of sender and receiver. Also requires providing confidentiality of principals' identities or their communication relationship.

Pseudonyms Fictitious names as a lightweight anonymity mechanism. E.g. blind ads in newspapers, stage names, email addresses, ...

Recipient anonymity: broadcasting Broadcast message to all group members. Addressee must be identified by an attribute, visible to her while invisible to others.

E.g. encrypt attribute with public key of addressee. Recipients of broadcast decrypt all such attributes received.

Drawbacks: Scalability, denial of service, ...

Sender anonymity: proxies Packets, e.g. HTTP requests, are anonymized by a proxy.

Weaknesses: Single point of failure: proxy knows everything and needs high availability. Traffic analysis is also possible: message headers identify recipients and packet routes can be traced.

Cascaded proxies with encryption Assume client has proxy's public key. Client encrypts message M for proxy along with server address S. Generalize to a chain of proxies with cascaded encryption.

Improvement: each proxy only knows previous/next hop. But traffic analysis is still possible.

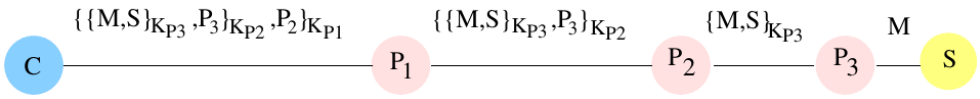

Figure 49: Proxy chain with cascaded encryption

12.2.1. Mix networks

Mix Server processing messages M destined for address A:

$$\left\{R_1, \{R_0, M\}_{K_A}, A\right\}_{K_1} \longrightarrow \{R_0, M\}_{K_A}, A$$

K_i is the public key of mix i and the R_j are random strings (for randomised encryption).

To foil traffic analysis, a mix does additional things:

- **Hide message size**: messages are of uniform size, split and pad where necessary
- **Hide order of arrival**: output items in batches
- **Suppress repeated information**: filter duplicate messages
- **Sufficient traffic**: clients send/receive dummy messages

Message receipts A mix can (optionally) return a receipt Y for a received message:

$$Y = \left\{C, \left\{R_1, \{R_0, M\}_{K_A}, A\right\}_{K_1}\right\}_{K_1^{-1}}$$

where C is a large known constant. Using Y, a client can later proof that she sent a message $X = \{R_0, M\}_{K_A}, A$ by revealing X and R_1. Other can verify that $\{Y\}_{K_1} = C, \{R1, X\}_{K_1}$.

Mix networks [David Chaum, 1981]

Designed to provide unlinkability against a global Dolev-Yao attacker.

Sender chooses a random path of multiple mixes from different administrative domains. Works in cascade, with each mix "peeling" off the outermost "layer":

$$\left\{R_n, \left\{R_{n-1}, ..., \left\{R_1, \{R_0, M\}_{K_A}, A\right\}_{K_1} ..., A_{n-2}\right\}_{K_{n-1}}, A_{n-1}\right\}_{K_n}$$

where mixes have keys $K_n, ..., K_1$ and addresses $A_n, ..., A_1$.

Untraceable return addresses To reply to an anonymous sender x:

1. Sender sends her "return address": $\{R_1, A_x\}_{K_1}, K_x$
   K_x is a fresh public key and A_x her real address.
2. Receiver sends reply M to mix: $\{R_1, A_x\}_{K_1}, \{R_0, M\}_{K_x}$
3. Mix transforms and sends to sender: $A_x, \left\{\{R_0, M\}_{K_x}\right\}_{R_1}$
   Note how R_1 is used as a symmetric key to mask input/output correlation across the mix.

Again, this generalises to a path of mixes using layering (see slide 40). We end up with

$$A_x, \left\{\left\{\left\{\{R_0, M\}_{K_x}\right\}_{R_1}\right\}_{R_2} \ldots\right\}_{R_n}$$

Pros and cons

- ⊕ No input/output correlation
- ⊕ Only a single mix per path needs to be honest
- ⊕ Anonymity set = entire network, given enough (dummy) traffic
- ⊖ Higher latency, possibly lower bandwidth
- ⊖ Overhead of multiple encryption and dummy messages

Mixes in practise Early examples already in 2000 (Zeroknowledge Systems Freedom Network). Today, the *Tor project* implements a mix variant called *onion routing*. It uses an entry, relay and an exit node. Tor has realtime guarantees since messages are not buffered indefinitely (when mixes lack dummy traffic), though it is still pretty slow for daily use. Since it operates at the TCP level, applications above can use it. However, the lack of dummy traffic makes it vulnerable to traffic analysis.

Crowds [Reiter & Rubin, 1998]

- Peer-oriented: each client is also a server
- *Jondo* (from "John Doe"): software that each client runs
- *Blender*: collects and distributes information about jondos in the network
- Messages are sent on random paths and encrypted between jondos using symmetric keys distributed by the blender. Each jondo forwards a message with probability $p_f > 0.5$ to another jondo, and with probability $1 - p_f$ to the destination web server.

12.3. Data protection

Personally Identifying Information PII can be linked to other data. But also non-PII data may be worthy of protection: intellectual property, business/military data, etc.

General Data Protection Regulation GDPR Regulates data processing for services accessible from the EEA – thus also relevant for non-EEA companies! It has the notions of *data subjects*, *data controllers* and *data processors*.

Data subjects have more rights under GDPR (c.f. Art. 12-23):

- Transparency: for which purpose is data collected?
- Access control and security of collected data
- Receive copy of personal data
- Rectification of inaccurate personal data
- Erasure ("right to be forgotten")
- Restriction of processing
- Data portability

Questions companies need to answer:

- What data is collected and processed? Where and stored for how long?
- For which purposes?
- How are data subjects' rights protected?
- How to detect, react and report a data breach? Possible consequences?

Complaints can be filed with the local supervisory authority. Fines can be up to max{20 mio. €, 4% of global annual turnover}.

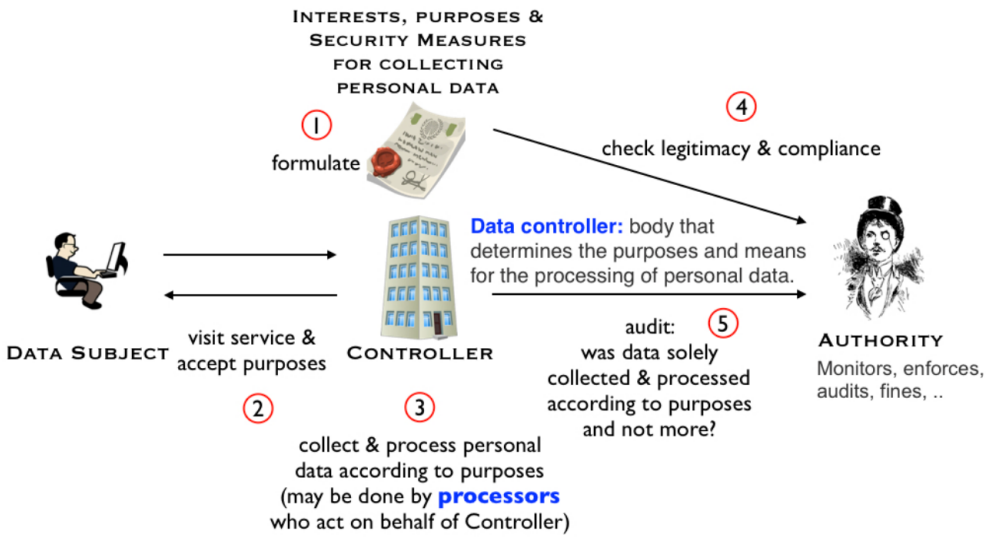

Figure 50: GDPR overview

Data protection policy Specifies *how* data may be used, under what *conditions* and what *obligations* this entails.

One machine-friendly (though discontinued) approach is *Platform for Privacy Preferences P3P* by the W3C. Website can specify their policy in XML, allowing browsers to compare it against and react to user preferences. Other approaches exist, but non is widely adopted.

Enforcing polices The technical challenge is to build preventive control mechanisms, in order to not rely on manual human auditing.

System model: Actors can be *data providers* (aka *servers*), *data consumers* (aka *clients*) or both. Each data item has a *data owner*.

Requirements can arise from the data provider, data owner, law or contracts. They are time dependent: *Provisions* express conditions in the past and present whereas *obligations*[34] express conditions on the future. The latter is hard to enforce.

Enforcement: A requirement is enforceable if there exists a mechanism that ensures that all executions of the system satisfy the requirement. Enforcing provisions requires *access control*[35], and obligations require *usage control*.

Mechanisms for server-side enforcement exist. Client-side restrictions (against careless or malicious client) are harder: *Controllability* of clients is rare (c.f. DRM). We can attempt *observability* of whether obligations are met, e.g. by auditing or water-marking. If they are not, we can issue *compensating actions* (c.f. speed cameras).

[34] For example: duration to store data for, purpose, notification of the owner when using data, secure storage, ...
[35] Not so trivial with GDPR: how do you control purpose of access?

13. E-Voting

Types of voting systems

- Paper
- Mechanical (e.g. using marbles)
- Direct-recording machines (input vote on touch screen)
- Optical scan voting systems (reads papers + automates tallying)
- Internet voting

Advantages of e-voting include lower administrative cost, increased accessibility, increased usability (e.g. warn user about invalid votes), faster tallying. The main challenge is maintaining vote secrecy while also having verifiability.

13.1. Modelling voting systems

Roles

- *Election authority* = *Registration tellers* (generate voter credentials) ∪ *Tabulation tellers* (collect + count votes)
- *Voters*
- *Devices*
- *Auditor:* checks for malicious behaviour
- *Bulletin board:* public data storage, e.g. containing election result

Adversary model We assume a Dolev-Yao adversary.

Trust assumptions: devices can be compromised, but not voters. Authority can manipulate election (i.e. not trusted for verifiability), but is trusted for privacy.

Channel assumptions: assume an anonymous channel (one where traffic analysis is impossible) for sending the vote (out-of-scope for this, e.g. use Tor). Assume that registration happens on an authentic and confidential channel, e.g. by physical mail.

However, reality limits what formal protocol verification can cover: if the adversary can always be physically present, she can learn all voter actions (e.g. if there is no voting both). She may also exert pressure on entire groups, leading to indirect coercion.

Security properties Because the possible choices are public, we need stronger properties than simply confidentiality and integrity. E.g. consider the case where we have only one voter, thus the election outcome leaks what the voter voted. Broadly speaking, we want vote secrecy as well as verifiability in order to prevent corruption. Later we see that there exist schemes to realise both these seemingly conflicting goals.

Vote privacy: the adversary cannot link Alice's vote to Alice, that is she cannot distinguish the situations "Alice votes Yes" and "Alice votes No" (cf. unlinkability).

Receipt-freeness: a voter cannot prove (using a receipt) to the adversary how she voted. Stronger than privacy, requires voter *willingness* to cooperate.

Coercion resistance: Even when the adversary can actively give inputs to the voter, the voter cannot prove how she voted. Stronger than receipt-freeness.

Availability/no forced abstention: an adversary cannot prevent a voter from casting a vote. Sometimes seen as a subset of coercion resistance.

Everlasting privacy: vote privacy is ensured in the long term, even when computationally secure schemes are broken.

Individual verifiability: each voter can verify that her own vote is recorded as cast.

Universal verifiability: everyone can verify that the recorded votes are tallied (counted) correctly.

End-to-end verifiability: union of previous two. Everyone's vote is verifiably in the final result.

Eligibility verifiability: only registered voters can vote, and at most once.

Dispute resolution: when a voter detects manipulation, she can convince others that the authority is dishonest, yet the authority cannot be falsely convicted. Required in addition to verifiability, since potentially only a single voter is able to notice a manipulation on her vote.

13.2. Code voting

Process

1. Each voter receives a *ballot* with unique, random *vote codes*, with the corresponding candidates listed in random order.
   Note that knowledge of the specific ballot is private to the authority and the user.
2. Voters cast vote by sending in vote code of chosen candidate.
3. Authority matches (voter, vote code) to candidates and tallies votes.
4. Authority publishes tally **and** vote codes (lexicographically sorted) on public bulletin board.

vote code	candidate
1a	Asterix
1b	Obelix

vote code	candidate	
1a	Asterix	x
1b	Obelix	
2a	Obelix	
2b	Asterix	x
3a	Asterix	
3b	Obelix	x

vote code
1a
2b
3b

Tally = 2x Asterix, 1x Obelix

Table 3: Example ballot (left), authority table (middle) and bulletin board (right)

Individual verifiability: a voter can easily check whether the vote code she send in appears on the bulletin board, i.e. is counted. To check whether it is counted for the correct party, we need ballot auditing, see below.

Universal verifiability is harder, since we must not reveal the vote code–candidate correspondence (to preserve vote secrecy).

Cut & Choose Inspired by fairly dividing some cake: one party cuts it, then the other chooses a piece.

1. Authority shuffles authority table and reveals it, concealing "vote code" and "candidate" columns
2. Auditor chooses – with probability 0.5 – either of the two columns to be revealed. If the first is revealed, the auditor can check the vote codes on the bulletin board. If the latter is revealed, the auditor can check the tally.
3. Rinse and repeat. Shuffling ensures no information is leaked on the side.

Thus the authority can either only manipulate a small number of votes, or the manipulation will be detected with high probability.

Ballot audit Addresses the problem that the authority could manipulate the rows on the ballot, compared to what it puts into the authority table. E.g. it puts (1a, Asterix) on the ballot, but records (1a, Obelix), swapping the candidate ordering. This manipulates the tally, while going undetected by the external auditor.

1. Voters receive two ballots
2. Voter randomly chooses which ballot to use for voting and which for auditing
3. Authority reveals auditing ballots of all voters on the bulletin board

Now voters can audit the content of their second, unused ballot. Again, the authority can either not manipulate many ballots, or it is detected with high probability.

Therefore code voting achieves both end-to-end verifiability and vote privacy. However, it is not receipt-free – a voter can reveal his ballot.

13.3. Random-sample voting

Idea Only a randomly choosen subset of the electorate votes, allowing voters to vote less often and thus spent more time researching their decisions. Below, we discuss *Alethea*[36], an example Random Sample Voting Protocol developed at ETH Zurich. It consists of two phases: selection and voting.

Setup See Figure 51. Each voter has a secure device D, which can perform cryptographic operations for her. Votes are cast via an insecure platform P.

[36] https://inf.ethz.ch/personal/basin/pubs/csf18.pdf

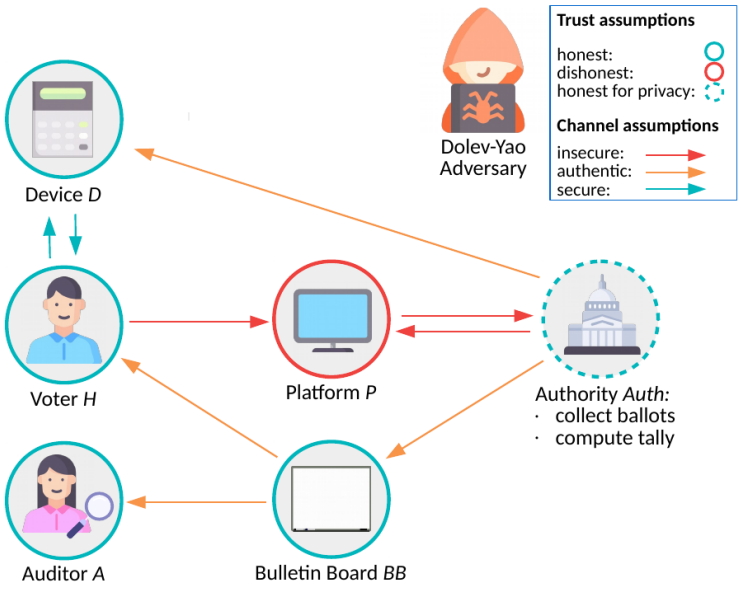

Figure 51: Alethea setup and assumptions

Selection phase

1. *Auth* publishes a commitment to a future event, that will happen and cannot be influenced (here: sampling lottery, election).
2. *Auth* generates fresh signing keys sk_D for all voters and their D. The corresponding "verification key" servers as the voter's pseudonym.
3. *Auth* publishes all pseudonyms on BB.
4. Lottery event: *Auth* computes sample group based on random seed committed to earlier.

Voters H can check individually whether their pseudonym appears on the bulletin board BB (individual verifiability). The sampling step can be checked by auditor A (universal verifiability). Furthermore, we have anonymity of the sampling group members.

Voting phase

1. H decides on a candidate.
2. D encrypts decision under *Auth*'s public key and signs with sk_D. Sends result to *Auth*.
3. *Auth* keeps ballot if signature is in sampling group.
4. *Auth* decrypts and shuffles votes (to remove input-output correlation, c.f. mix networks).
5. *Auth* creates a zero-knowledge proof of **plaintext equivalence** (of the input and the output).
6. *Auth* publishes set of received (encrypted) ballots, set of shuffled + decrypted votes and the ZKP on BB.

Voters H can check individually whether their ballot is on the bulletin board BB (individual verifiability). The auditor can verify the tally on BB, using ZKP to check that the set of plaintext votes were indeed computed from the set of encrypted votes (universal verifiability). This gives end-to-end verifiability. Under the assumption that users to not give away their devices, we have receipt-freeness (thgoebel: but wouldn't giving away the device be exactly such a willful cooperation action???).

Part III.
Appendix

A. Imprint

This document closely follows the lecture slides of the *Information Security* lecture in the spring semester 2020 at ETH Zurich. Our contribution to this is editing the whole lot and refactoring even more so that it may fit the "lecture summary" style. However, basically all graphics are copy & pasted from the slides. If you don't want yours here, please contact us and we will remove them.

Otherwise, our part of the work is published as CC BY-NC-SA.

B. Notation

Below is a non-exhaustive list of notations and symbols which are used throughout this document and which the reader may not have seen before:

- $||$ — concatenation operator
- $\oplus$ — xor operator
- $x \leftarrow G$ — x is chosen from group G
- PPT — probabilistic polynomial time

B.1. Message construct notation

Message	$\{M\}$
Name	A, B or *Alice*, *Bob*
Asymmetric keys	A has public key K_A and private key K_A^{-1}
Symmetric keys	K_{AB} shared by A and B
Encryption	asymmetric $\{M\}_{K_A}$ or symmetric $\{M\}_{K_{AB}}$
Signing	$\{M\}_{K_A^{-1}}$
Nonces	N_A ("number used once", freshly created by A)
Timestamps	T (eg expiration)

C. Reading Tamarin Graphs

Actions of the same agent are coloured with the same shade of green.

The boxes mirror a labeled multiset rewriting rule $l \xrightarrow{a} r$. Thus the three rows, top to bottom, mean:

1. state fact before rewriting
2. action fact/event
3. state fact after rewriting

Arrows have the following colour-coding:

- grey $\longrightarrow$ using persistent facts
- black $\longrightarrow$ using linear facts
- orange $\longrightarrow$ sending a message into to the network, i.e. to the adversary
- dotted $\longrightarrow$ attacker using a rule

Other syntactical things to note:

- \$ – prefixes public values (e.g. agent names, state facts.
  Notabene: recipients of messages containing values prefixed with \$ do check whether the value is indeed public.
- ! – prefixes persistent facts
- $\sim$ – prefixes fresh facts

Exam relevance:
It is sufficient to be able to read a Tamarin graph. You neither need be able to write Tamarin theory from scratch, nor to know any detailed syntax – but you are expected to be familiar with the Dolev-Yao attacker model and be able to reason about security protocols and their execution. See past exams for examples.