Applied Cryptography

thgoebel@ethz.ch

ETH Zürich, FS 2021

This documents is a **short** summary for the course *Applied Cryptography* at ETH Zurich. It is intended as a document for quick lookup, e.g. during revision, and as such does not replace reading the slides or a proper book.

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

Contents

List of Figures

Credits: images are generally taken from the lecture slides.

1 Symmetric Cryptography

One-time pad Plaintext p, key k such that $|p| = |k|$. Ciphertext $c = p \oplus k$.

If k u.a.r. and only used once then the OTP is **perfectly secure**, i.e. $\Pr[P = p|C = c] = \Pr[P = p]$.

Note: keys can re-occur (as a result of random sampling) but they must not be re-used (i.e. the adversary must not be aware that the same key is used).

Issues: same lengths, key distribution, single use.

1.1 Block Ciphers

Block cipher A block cipher with key length k and block size n consists of two efficiently computable permutations[1]:

$$E : \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^n \quad D : \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^n$$

such that for all keys K D_K is the inverse of E_K (where we write E_K short for $E(K, \cdot)$).

Security notions Known plaintext attack, chosen plaintext attack, chosen ciphertext attack. Exhaustive key search on (P, C) pairs – no attack should be better, else we throw the cipher away.

Pseudo-randomness

- Adversary $\mathcal{A}$ interacts either with block cipher (E_K, D_K) or a truly random permutation (Π, Π^{-1}).
- A block cipher is called a **pseudo-random permutation PRP** if no efficient[2] $\mathcal{A}$ can tell the difference between E_K and Π (no access to the inverse).
- A block cipher is called a **strong-PRP** if no efficient $\mathcal{A}$ can tell the difference between (E_K, D_K) and (Π, Π^{-1}).

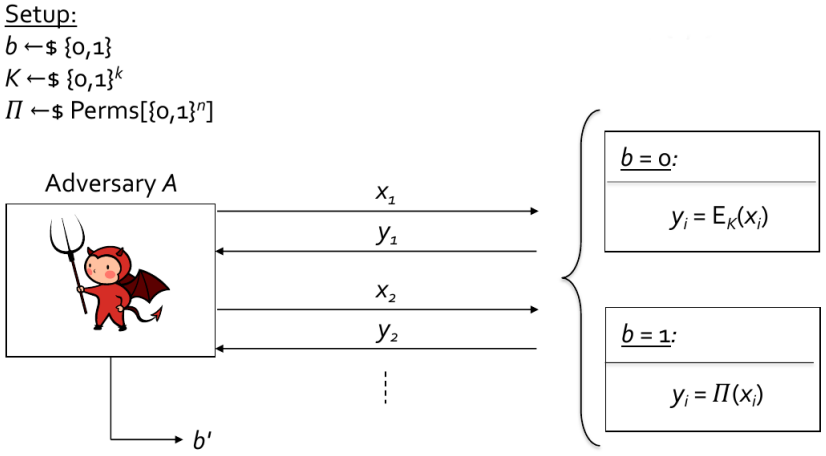

Figure 1: PRP game

[1] Encipher and decipher

[2] Quantified by runtime + number of oracle queries.

The advantage is defined as:

$$\mathbf{Adv}_E^{PRP}(\mathcal{A}) = 2 \cdot \left| \Pr[\text{Game } \mathbf{PRP}(\mathcal{A}, E) \Rightarrow \text{true}] - \frac{1}{2} \right|$$

where the probability is over the randomness of $b, K, \Pi, \mathcal{A}$. Note that:

$$\Pr[\text{Game } \mathbf{PRP}(\mathcal{A}, E) \Rightarrow \text{true}] = \Pr[b' = b]$$

Also see the Advantage Rewriting Lemma (1.2).

Constructing block ciphers In general: keyed round function that is repeated many times.

- Feistel cipher: halved blocks crossing back and forth, e.g. DES
- Substitution-permutation network: confusion + diffusion, e.g. AES

Electronic Code Book (ECB) mode Same plaintext always maps to the same ciphertext (deterministic). Thus serious leakage, don't use.

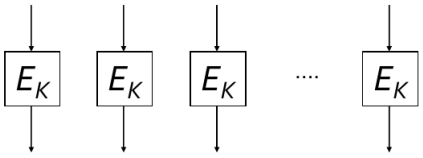

Figure 2: ECB mode

Cipher Block Chaining (CBC) mode Use u.a.r. IV/previous ciphertext block to randomise encryption.

A bit flip in C_i completely scrambles/randomises P_i and flips the same bit in P_{i+1}.

Caveats: non-random IV, padding oracle attack, ciphertext block collisions (after using the same key for $2^{n/2}$ blocks by the birthday bound).

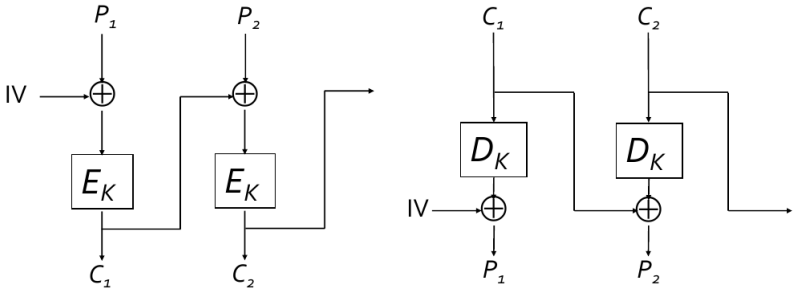

Figure 3: CBC mode (left: encipher, right: decipher)

Counter (CTR) mode Incrementing counter is encrypted with block cipher to produce a pseudorandom value to xor the plaintext block with.

Effectively a stream cipher producing OTP keys. E_K does not even need to be invertible. No padding needed, can just truncate the last block. A bit flip it C_i flips the same bit in P_i.

Caveats: counter must not repeat/wrap around (else xor of ciphertexts = xor of plaintexts).

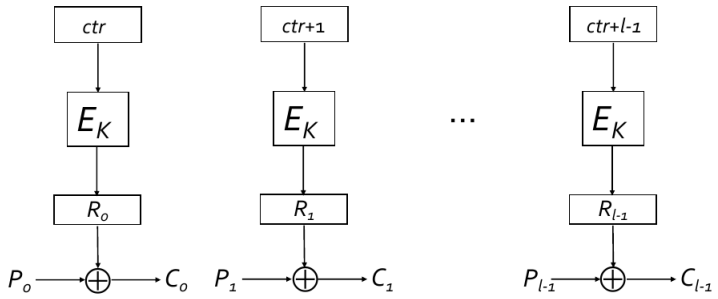

Figure 4: CTR mode

1.2 Symmetric Encryption

Symmetric Encryption Scheme is a triple $SE = (KGen, Enc, Dec)$. We have key space $\mathcal{K} = \{0,1\}^k$, message space $\mathcal{M} = \{0,1\}^{*3}$ and ciphertext space $\mathcal{C} = \{0,1\}^*$. For correctness, we have $Dec_K(Enc_K(m)) = m$.

IND-CPA Security Informally: computational version of perfect security – an efficient adversary cannot compute anything useful from a ciphertext (e.g. hide every bit of the plaintext). Equivalent to *semantic security*.

Formally: For any efficient adversary $\mathcal{A}$, given the encryption of one of two equal-length messages of its choice, $\mathcal{A}$ is unable to distinguish which one of the two messages was encrypted.

In the security game, $\mathcal{A}$ gets access to a *Left-or-Right encryption oracle*. The advantage of $\mathcal{A}$ is:

$$\mathbf{Adv}_{SE}^{IND-CPA}(\mathcal{A}) = 2 \cdot \left| \Pr[\text{Game } \mathbf{IND\text{-}CPA}(\mathcal{A}, SE) \Rightarrow \text{true}] - \frac{1}{2} \right|$$

Notes: Deterministic schemes **cannot** be IND-CPA secure (why?[4]). CBC and CTR mode (if used properly) can be proven to be IND-CPA secure (assuming that Enc is a PRP-secure block cipher).

Caveats: No integrity. Says nothing about messages of non-equal length. No chosen ciphertexts.

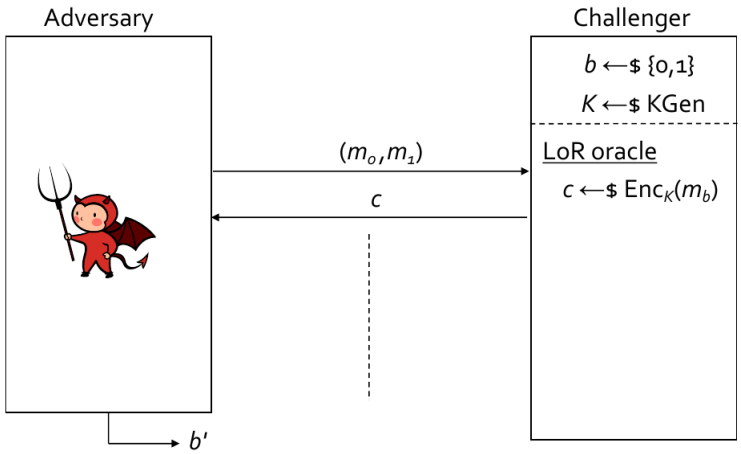

Figure 5: IND-CPA game

[3] In reality we might have a maximum plaintext length.
[4] The adversary can first query (m_0, m_0) to learn c_0, then query (m_0, m_1) and thus win with high probability.

Advantage Rewriting Lemma Let b be a uniformly random bit and b' the output of some algorithm. Then:

$$\begin{aligned} 2\left|\Pr[b'=b]-\frac{1}{2}\right| &= \left|\Pr[b'=1|b=1]-\Pr[b'=1|b=0]\right| \\ &= \left|\Pr[b'=0|b=0]-\Pr[b'=0|b=1]\right| \end{aligned}$$

Difference Lemma Let Z, W_1, W_2 be events. If

$$(W_1 \wedge \neg Z) \text{ occurs if and only if } (W_2 \wedge \neg Z) \text{ occurs}$$

then

$$\left|\Pr[W_2]-\Pr[W_1]\right| \leq \Pr[Z]$$

In practice: Z is a bad event that rarely happens, W_1, W_2 are when $\mathcal{A}$ wins in security games G_1, G_2. Useful for *game hopping* proofs.

PRP-PRF Switching Lemma Let E be a block cipher. Then for any algorithm $\mathcal{A}$ making q queries:

$$\left|\mathbf{Adv}_E^{PRP}(\mathcal{A})-\mathbf{Adv}_E^{PRF}(\mathcal{A})\right| \leq \frac{q^2}{2^{n+1}}$$

1.3 Attacks

Padding Added before encryption to expand plaintext to a multiple of the block size, i.e. $pad(\cdot) : \{0,1\}^* \mapsto \{\{0,1\}^n\}^*$. Must be expanding (why?) and efficiently computable. May be either randomised or deterministic.
E.g. TLS padding: appends $t+1$ bytes of value t.

Problem: adversary can detect padding errors (explicit error messages, logs, timing differences).

Padding Oracle Given a ciphertext C, returns whether the padding of the decryption is valid or invalid. Leaks 1 bit of information. Kind of a chosen ciphertext attack, thus not covered by IND-CPA security! Main problem: no ciphertext integrity.

In practice: $\mathcal{A}$ needs 128 oracle queries on average to recover one plaintext byte. Repeat for all bytes and all blocks for full plaintext recovery. Additional practical details to consider. For TLS attacks, see Lucky 13 and POODLE.

Predictable IVs Random IVs are not just theoretically relevant for IND-CPA security of CBC mode. Consider the following steps (see Figure 7):

1. $\mathcal{A}$ queries LoR oracle with (P_0, P_1)
2. Oracle responds with $C = C_0||C_1$ where $C_1 = E_K(P_b \oplus C_0)$
3. $\mathcal{A}$ predicts next $IV = C_0'$
4. $\mathcal{A}$ queries $P_0 \oplus C_0 \oplus C_0'$
5. $\implies b = 0 \iff P_b = P_0 \iff C_1 = C_1' \implies$ breaks IND-CPA security

In practice: systems may use *IV chaining* (use the last ciphertext as the next IV, in order to avoid having to sample new randomness). E.g.: SSLv3, TLS 1.0, SSH in CBC mode. See also the BEAST attack on TLS.

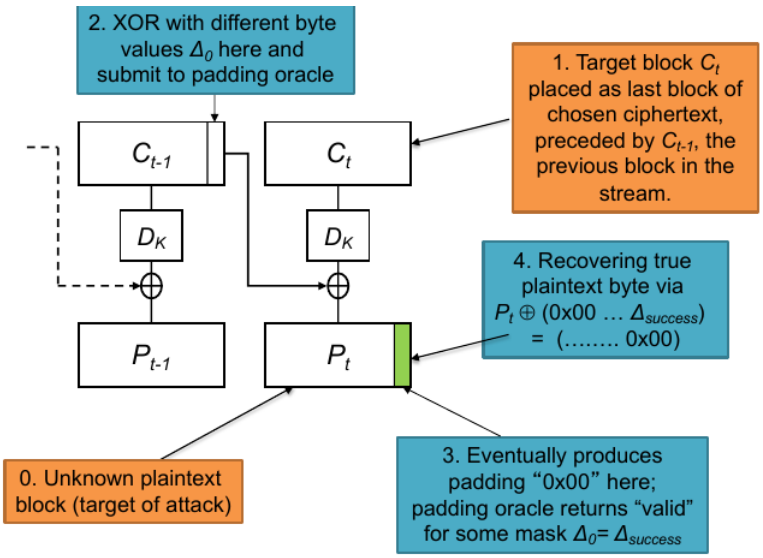

Figure 6: Padding oracle attack on CBC mode

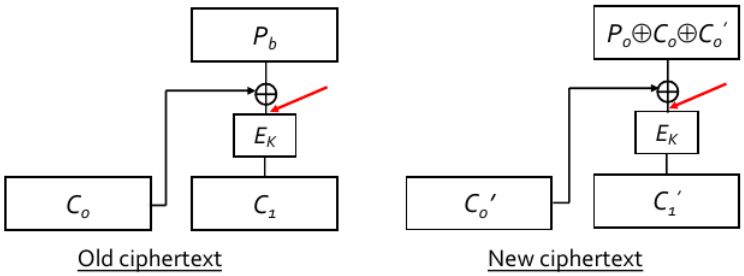

Figure 7: Attack on predictable IVs in CBC mode

1.4 Hash Functions

Cryptographic Hash Function An efficiently computable function $H : \{0,1\}^* \mapsto \{0,1\}^n$ that maps arbitrary-length inputs to fixed-length outputs ("*message digests*"). Not keyed!

Applications: MACs, signatures, key derivation, commitments.

Random oracle model Model under which hash functions are assumed to produced outputs that are uniformly random distributed. I.e. a hash function could be modelled by a random oracle. Very strong assumption!

Birthday paradox When drawing elements at random from a set of size s then after $\sqrt{s}$ trials we expect a collision with 39% probability. We quickly get to 99% with an additional constant factor.

Security goals Primary goals:

- *Pre-image resistance* (one-wayness): Given h, it is infeasible to find an m such that $H(m) = h$.
- *Second pre-image resistance*: Given m_1, it is infeasible to find an m_2 such that $H(m_1) = H(m_2)$.
- *Collision resistance*: It is infeasible to find any $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$.

Secondary goals:

- *Near-collision resistance*: It is infeasible to find any $m_1 \neq m_2$ such that $H(m_1) \approx H(m_2)$ (e.g. hashes agree on most bits).
- *Partial pre-image resistance 1*: Given $H(m)$, it is infeasible to recover any partial information about m.
- *Partial pre-image resistance 2*: Given a target string t with $|t| = l$ it is infeasible to find an m such $H(m) = t||x$ (faster than with 2^l brute-force hash evaluations).

CR adversary:
Cannot quantify security over all efficient $\mathcal{A}$ (collisions exist, $\mathcal{A}$ can hardcode one). Instead, define "(t, ε)-CR adversaries", running in time t and with $\mathbf{Adv}_H^{CR}(\mathcal{A}) = \varepsilon$. Build reductions from there.

Relationships:

- Collision resistance $\implies$ second pre-image resistance
- Maybe: Collision resistance $\implies$ pre-image resistance – depending on how you define pre-image resistance (sampling from the domain or from the range?)

Generic attacks (in the ROM):
(Second) pre-image resistance: 2^n evaluations.
Collision resistance: $2^{n/2}$ evaluations! (by the birthday paradox)
$\implies$ e.g. SHA-1 with 160-bit outputs only achieves 80-bit security

Merkle-Damgård iterated hashing Constructs a hash function H from a *compression function* $h : \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^n$. Used e.g. in MD5, SHA-1, SHA-2.

Steps: pad message, split into k-bit chunks, repeatedly apply h to the chunks and IV/chaining values (see Figure 8).

Padding scheme:

$$m' = pad(m) = m||10^t||[len(m)]_k$$

where $[len(m)]_k$ is the k-bit encoding of the message length[5] and $0 \leq t < k$ is minimal.

Theorem: If h is collision resistant, then so is H.[6]

Length extension attack: Given $H(m)$ (but not m!) once can compute a valid hash $y = H(pad(m)||m'')$. This is problematic e.g. when construction MACs or KDFs from a hash function.

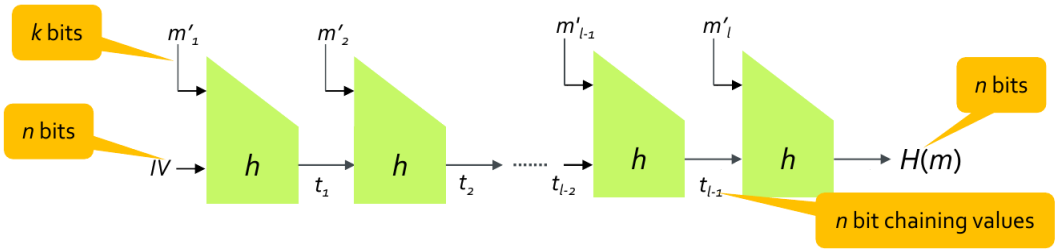

Figure 8: Merkle-Damgård transform

[5]This limits the maximum length of a message that can be hashed to $2^k - 1$.
[6]Note that the choice of padding scheme matters for the proof! Also note that the two IVs must be the same for a *full collision*. The much weaker form of two colliding messages for different IVs is called a *freestart collision*.

Constructing compression functions from block ciphers E.g. Davies-Meyer, Matyas-Meyer-Oseas, Miyaguchi-Preneel constructions. Use message as the key (need fast rekeying!) and (some variation of) chaining value as "plaintext" input.

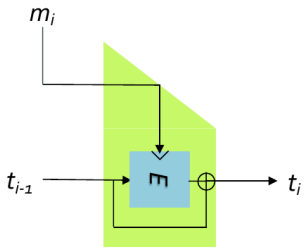

Figure 9: Davies-Meyer construction

Sponge construction Different design approach. Centered around 2 phases: absorbing + squeezing. Key ideas: giant bit permutation F, not inputting/outputting entire state. Variable length output.

E.g. used in SHA-3/Keccak: SHA-3$(m) = out_1||out_2||out_3||....$

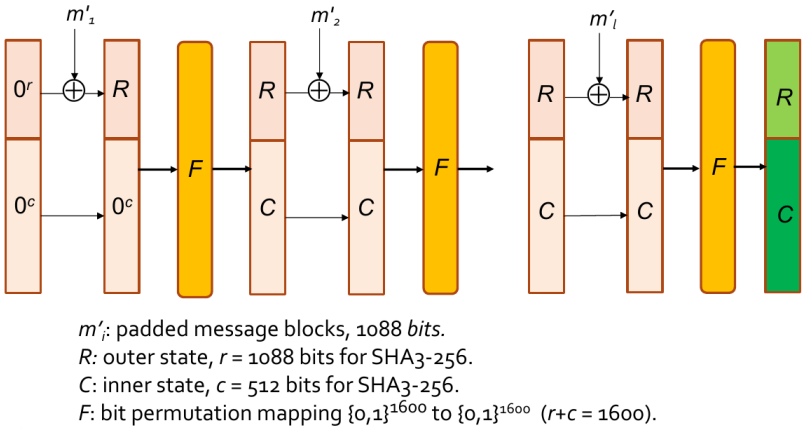

Figure 10: Sponge construction: absorbing

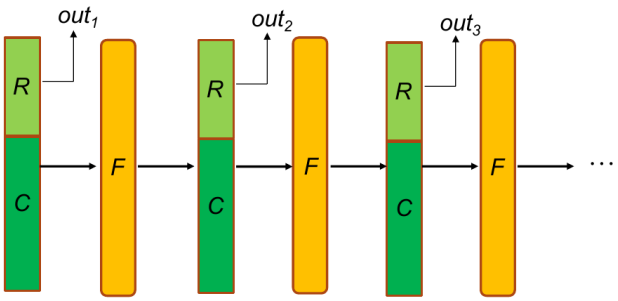

Figure 11: Sponge construction: squeezing

1.5 Message Authentication Codes MACs

Motivation Provide *integrity* and *data origin authentication*, i.e. no attacker should be able to modify or forge messages (unforgeability).

They do not provide confidentiality or secure against message deletion, replay, reordering, or reflection (if the key is used in both directions).

Definition (MAC scheme) A MAC scheme with key length k and tag length t consists of three efficient algorithms:

$$\begin{aligned} KGen: &\quad \{\} \mapsto \{0,1\}^k \\ Tag: &\quad \{0,1\}^k \times \{0,1\}^* \mapsto \{0,1\}^t \\ Vfy: &\quad \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^t \mapsto \{0,1\} \end{aligned}$$

For correctness[7], we require that

$$\forall K, m \,.\, Vfy(K, M, \tau) = 1 \text{ where } \tau = Tag(K, m)$$

Vfy and Tag may be randomised. Note that if they are deterministic and Vfy internally (re)computes $Tag(K, m) \stackrel{?}{=} \tau$, then the scheme has *unique tags*.

MAC Security Informally: it is hard for an attacker to forge a valid message-tag pair (m', τ').

Security game: $\mathcal{A}$ gets access to a tag oracle and a verify oracle.

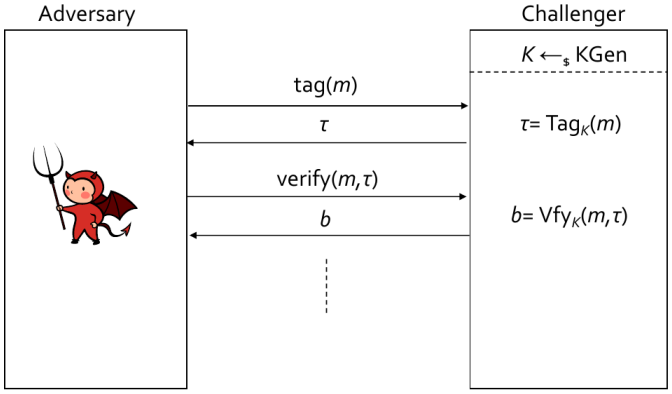

Figure 12: MAC game

Weak unforgeability under chosen message attack (WUF-CMA) $\mathcal{A}$ wins if it submitted some (m^*, τ^*) to the verify oracle such that $Vfy(m^*, \tau^*) = 1$ and no query $Tag(m^*)$ was made.

Strong unforgeability under chosen message attack (SUF-CMA) $\mathcal{A}$ wins if it submitted some (m^*, τ^*) to the verify oracle such that $Vfy(m^*, \tau^*) = 1$ and no query $Tag(m^*)$ with response τ^* was made.

Note that:

- A SUF-CMA adversary can also win by coming up with a new tag for an old message. We make the adversary "stronger" by relaxing the winning condition.
- SUF-CMA $\implies$ WUF-CMA (somewhat counter-intuitively)

[7] A functionality, not a security requirement.

- For deterministic MAC schemes that build Vfy from Tag, the two are equal (why?[8]).
- More formally: $(q_t, q_v, t, \varepsilon)$-W/SUF-CMA security

Generic attacks: guess (m, τ) pairs, guess the key, few tag queries + exhaustive key search.

MACs from PRFs Construction of $MAC(F)$: Let $F : \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^t$ be a PRF.

$$\begin{aligned} KGen() : &\quad K \leftarrow \$\ \{0,1\}^k \\ Tag(K,m) : &\quad \tau \leftarrow F(K,m) \\ Vfy(K,m,\tau) : &\quad \tau' \leftarrow F(K,m);\ \text{return } \tau' == \tau \end{aligned}$$

Theorem: If F is a PRF then $MAC(F)$ is a SUF-CMA.

Domain extension with CR hashing Motivation: build a MAC for a larger input domain $\mathcal{X}'$ from a MAC with fixed domain $\mathcal{X}$ (to handle more than just block-sized messages).

Construction of $HtMAC$: Let $MAC = (KGen, Tag, Vfy)$ be a MAC and let $H : \mathcal{X}' \mapsto \mathcal{X}$ be a hash function.

$$\begin{aligned} Tag'(K,m) : &\quad Tag(K, H(m)) \\ Vfy'(K,m,\tau) : &\quad Vfy(K, H(m), \tau) \end{aligned}$$

Theorem: If MAC is SUF-CMA secure and H is collision resistant then $HtMAC$ is SUF-CMA.

Hash-based MAC HMAC Turn (the often fast) hash functions into a keyed primitive using a two-key nest to build a PRF (and thus a MAC):

$$F_{nest}((K_1, K_2), m) = H(K_2 || H(K_1 || m))$$

Notes: Cannot simply prepend the key (length extension attacks). Also cannot simply append the key (offline collision attacks). In practice, derive two keys from one key using an inner and outer mask. Implementations often use an initialise-update-finalise interface (which can open timing attack vectors).

HMAC is gradually supplanted by faster designs based on universal hashing (see below), yet it is still used in key derivation scenarios (that use PRFness).

Nonce A number used once. Does neither need to be secret, nor does it need to be unpredictable. But must never ever repeat. Reasoning: it is easier to only use a number once than it is to get a good source of randomness.

Nonce-based MAC NMAC Like normal MAC, but Tag and Vfy now also take a nonce $N \in \mathcal{N}$. In the security game, we require that nonces are distinct (e.g. enforced by the tag oracle).

[8] When tags are unique then no adversary can come up with another tag for a previously queried message. However, consider the following deterministic scheme that is WUF-CMA but not SUF-CMA:

$$Tag'(K,m) = 0 || Tag(K,m) \quad Vfy'(K,m,b||\tau) = Vfy(K,m,\tau)$$

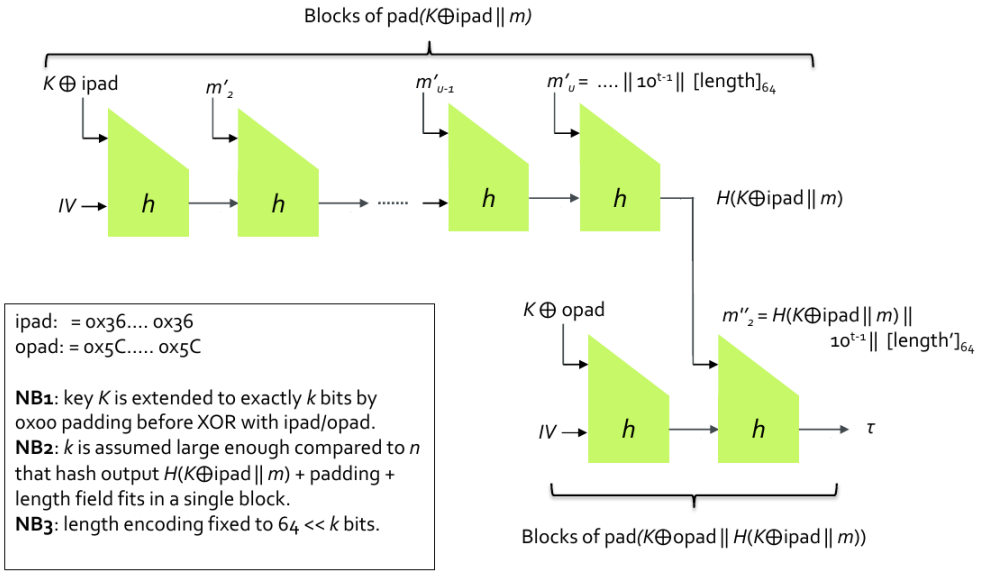

Figure 13: HMAC

Keyed hash function A keyed hash function H is a deterministic algorithm that takes a key and a message and produces a *digest* $t = H(K, m)$ over the digest space $\mathcal{T}$.

Universal hashing UHF Informally: selecting a hash function at random from a family of hash functions (the universal family).

Security game: Let H be a keyed hash function. Then:

1. Challenger sets $K \leftarrow \$ \{0,1\}^k$
2. Adversary outputs distinct (m_0, m_1)

$\mathcal{A}$ wins if $H(K, m_0) = H(K, m_1)$ and has advantage $\mathbf{Adv}_H^{UHF}(\mathcal{A})$.

H is an ε-*bounded universal hash function* if $\mathbf{Adv}_H^{UHF}(\mathcal{A}) \leq \varepsilon$ for all $\mathcal{A}$ (even unbounded ones).

Equivalent definition: H is an ε-UHF if $\forall m_0, m_1$. $\Pr[H(K, m_0) = H(K, m_1)] \leq \varepsilon$, where the probability is over the random choice of K.

Universal hash functions from polynomials Let $\mathcal{F}$ be a finite field (e.g. integers mod p or $GF(2^n)$) and let $\mathcal{K} = \mathcal{T} = \mathcal{F}$ and $\mathcal{M} = \mathcal{F}^{\leq l}$. Then we define:

$$H_{poly}(K, (a_1, \ldots, a_v)) = K^v + a_1 K^{v-1} + a_2 K^{v-2} + \cdots + a_{v-1} K + a_v$$

That is, H_{poly} is always a degree v polynomial (even if the message vector a is zero). It can efficiently be evaluated using finite field operations and Horner's rule.

Theorem: H_{poly} is an ε-UHF for $\varepsilon = \frac{l}{|\mathcal{F}|}$.

Difference unpredictable hashing DUHF Security game: Let H be a keyed hash function and let the digest space $\mathcal{T}$ have a group operation $+$ (with inverse $-$)[9]. Then:

1. Challenger sets $K \leftarrow\$ \{0,1\}^k$
2. Adversary outputs distinct $m_0, m_1 \in \mathcal{M}$ and $\delta \in \mathcal{T}$.

$\mathcal{A}$ wins if $H(K, m_0) - H(K, m_1) = \delta$ and has advantage $\mathbf{Adv}_H^{DUHF}(\mathcal{A})$.

H is an *ε-bounded distance unpredictable hash function* (ε-DUHF) if $\mathbf{Adv}_H^{DUHF}(\mathcal{A}) \leq \varepsilon$ for all $\mathcal{A}$ (even unbounded ones).

XOR universal hashing Special case of DUHF where we have XOR as the group operation. We define H_{xpoly} as follows:

$$\begin{aligned} H_{xpoly}(K, (a_1, \ldots, a_v)) &= K^{v+1} + a_1 K^v + a_2 K^{v-1} + \cdots + a_{v-1} K^2 + a_v K \\ &= K \cdot H_{poly}(K, (a_1, \ldots, a_v)) \end{aligned}$$

Theorem: H_{xpoly} is an ε-DUHF for $\varepsilon = \frac{l+1}{|\mathcal{F}|}$.

Carter-Wegman MACs Let H be an ε-DUHF and let F be a PRF. Then we define CW-MAC(F, H) as follows:

$$\begin{aligned} KGen() &: \quad (K_1, K_2) \leftarrow\$ \mathcal{K}_H \times \mathcal{K}_F \\ Tag((K_1, K_2), N, m) &: \quad \tau \leftarrow H(K_1, m) + F(K_2, N) \\ Vfy((K_1, K_2), N, m, \tau) &: \quad \tau' \leftarrow Tag((K_1, K_2), N, m);\ \text{return } \tau' == \tau \end{aligned}$$

Theorem: If H is an ε-DUHF and F a PRF then CW-MAC(F, H) is SUF-CMA secure.

In practice: e.g. GMAC (used in AES-GCM) instantiates F with AES and H with H_{xpoly} over $\mathcal{F} = GF(2^{128})$.

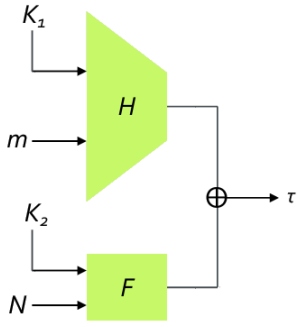

Figure 14: Carter-Wegman MAC tag algorithm

[9] E.g. addition mod n in $\mathbb{Z}_n$ or XOR in $\{0,1\}^n$.

1.6 Authenticated Encryption

Integrity of ciphertexts INT-CTXT Informally: adversary cannot forge new ciphertexts.

Security game: $\mathcal{A}$ can submit any m to an encryption oracle to obtain $c = Enc_K(m)$. At the end $\mathcal{A}$ submits one c^* to a try oracle.

$\mathcal{A}$ wins if 1) c^* is distinct from all previously seen c and 2) c^* decrypts to a $m^* \neq \perp$.

$\mathcal{A}$ has advantage $\mathbf{Adv}_{SE}^{INT-CTXT}(\mathcal{A})$.

Integrity of plaintexts INT-PTXT Informally: adversary cannot force a new plaintext to be accepted by the receiver.

Same as INT-CTXT but with an additional winning requirement:
$\mathcal{A}$ wins if ... 3) m^* is distinct from all previous queries m.

INT-CTXT $\implies$ INT-PTXT[10]

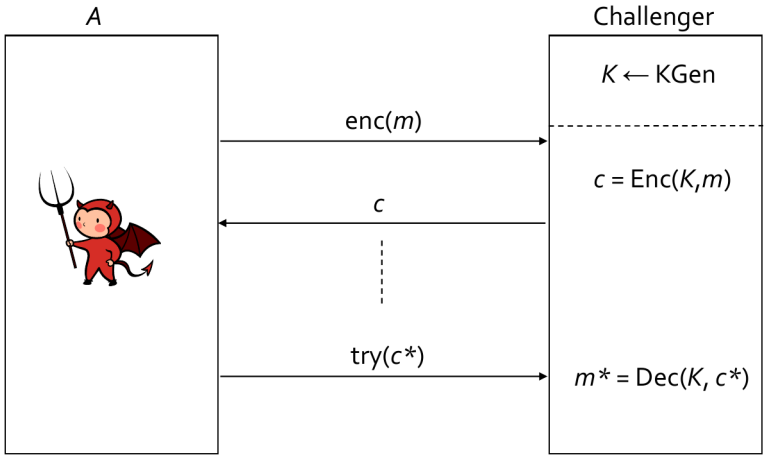

Figure 15: INT-CTXT + INT-PTXT game

IND-CCA Security (Indistinguishable under chosen ciphertext attack)

Same as IND-CPA (LoR oracle), with an additional decryption oracle.[11]

Authenticated Encryption AE An SE is said to be *AE secure* if it is IND-CPA secure and INT-CTXT secure.

AE $\implies$ IND-CCA[12]

Encrypt-and-MAC E&M Compute $c \leftarrow Enc_{KE}(m), \tau \leftarrow Tag_{KM}(m)$ and output $c' = c||\tau$.

NOT secure in general: If MAC is deterministic (e.g. a PRF) then trivial IND-CPA attack applies.

E.g. in SSH (in a stateful variant).

[10] Since due to SE correctness a new plaintext implies a new ciphertext.
[11] Trivially, the adversary is not allowed to submit ciphertexts received from the LoR oracle to the decryption oracle.
[12] Informal proof by case distinction: Either at some point the IND-CCA adversary queries a c that does not decrypt to $\perp$ (breaking INT-CTXT) or never, in which case the decryption oracle is useless (breaking IND-CPA).

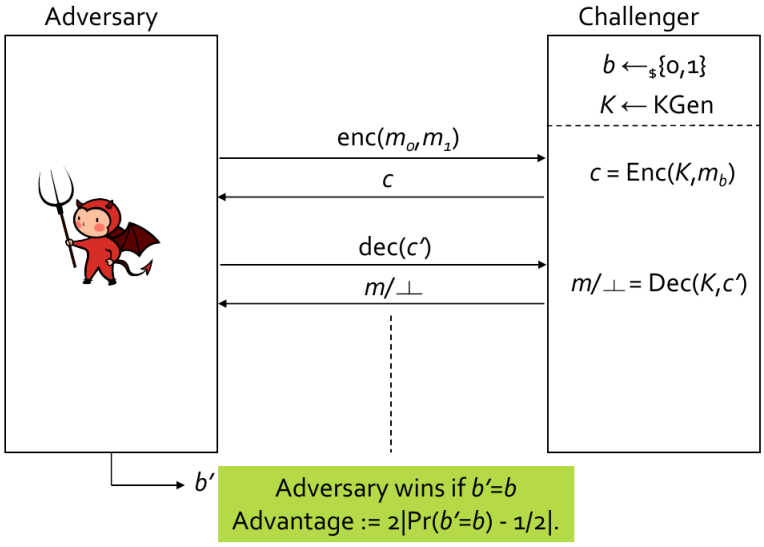

Figure 16: IND-CCA game

Mac-then-Encrypt MtE Compute $\tau \leftarrow Tag_{KM}(m)$ and output $c \leftarrow Enc_{KE}(m||\tau)$.

Can be made secure under specific instantiations, but unsafe in general – AVOID!

E.g. in TLS prior to 1.3.

Encrypt-then-Mac EtM Compute $c \leftarrow Enc_{KE}(m), \tau \leftarrow Tag_{KM}(c)$ and output $c' = c||\tau$.

AE secure under the assumption that SE is IND-CPA and MAC is SUF-CMA.

E.g. in IPSec ESP.

AE with Associated Data AEAD Informally: integrity-protect some data, provide confidentiality for the rest. E.g. AES-GCM, ChaCha20-Poly1305.

Both Enc and Dec also take the associated data AD as an additional input argument – but the ciphertext does not "contain" AD (instead AD is sent alongside, like the nonce N).
But: AD is not an output of the AEAD scheme (only c and τ are)!

IND-CPA game: adversary sends $(N, AD, (m_0, m_1))$ to LoR oracle.

INT-CTXT game: adversary sends (N, AD, m) to encryption oracle and (N^*, AD^*, c^*) to the try oracle.

EtM for AEAD:

$$c \leftarrow Enc_{KE}(m), \tau \leftarrow Tag_{KM}(len(AD)||AD||c);\ \text{output } c' = c||\tau$$

Note that tagging over the length is required to avoid mis-parsing between AD and c.

In practice: Enc and Tag may be nonce-based. If nonces implicitly use a counter, then the adversary cannot reorder or delete messages (and of course can't insert either due to INT-CTXT).

AES-GCM (Galois Counter Mode) EtM with AES in CTR mode and a CW-MAC (H_{xpoly} over $GF(2^{128})$). Single key, MAC key is derived: $K_M = AES(K, 0^{128})$. Nonces are usually 96 bit and reuse results in catastrophic failure (MAC key recovery), thus not *nonce-misuse resistant*. Counters for *Enc* are of the form $N||ctr$. Almost as fast as CTR mode, since UHF in MAC is fast.

2 Asymmetric Cryptography

Public Key Encryption (PKE) Scheme is a triple $PKE = (KGen, Enc, Dec)$.
$KGen$ generates a key pair $(sk, pk) \in \mathcal{SK} \times \mathcal{PK}$.
Enc takes a public key pk and a message $m \in \mathcal{M} \subseteq \{0,1\}^*$ and produces a ciphertext $c \in \mathcal{C} \subseteq \{0,1\}^*$.
Dec takes a private key sk and a ciphertext and produces a message or an error $\perp$.
For correctness, we require that $Dec_{sk}(Enc_{pk}(m)) = m$ (for all keypairs output by $KGen$).

Key distribution PKE translates the problem of securely distributing secret symmetric keys into distributing authentic public keys.

IND-CCA Security The adversary gets access to a LoR encryption and a decryption oracle and needs to distinguish whether the left or the right messages are being encrypted (see Figure 17). Compare this with the symmetric game (given in Figure 16).
IND-CPA security is defined the same, but without the decryption oracle.

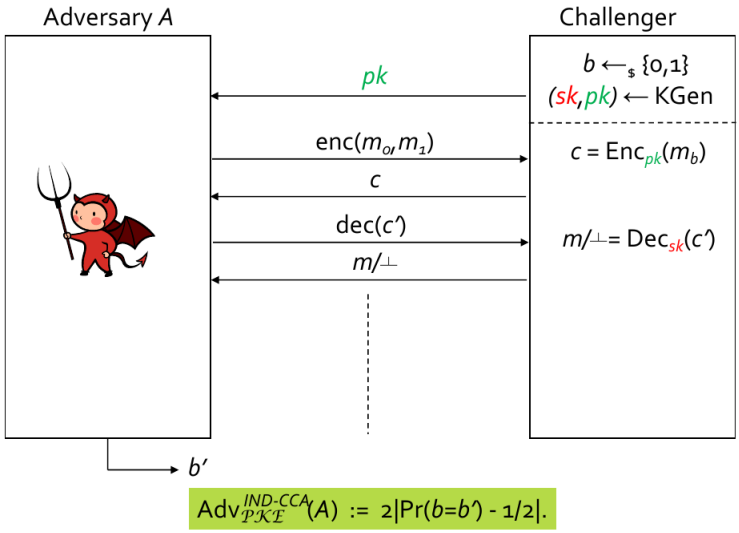

Figure 17: IND-CCA game for PKE

Integrity for PKE Defining integrity does not make sense in a PKE setting. Anybody can choose a message, encrypt it under the public key and can thus come up with a valid ciphertext that will be accepted by the receiver.

2.1 KEM/DEM Paradigm

Hybrid encryption Asymmetric crypto is expensive (e.g. due to large key sizes). Idea: encrypt the message symmetrically and encrypt the symmetric key using PKE.

Key Encapsulation Mechanism KEM is a triple $(KGen, Encap, Decap)$.
$KGen$ generates a random key pair (sk, pk).
$Encap$ takes a public key pk and generates a random key K, outputting the encapsulation c and the generated key: $(c, K) \in \mathcal{C} \times \mathcal{K}$.

Decap takes a private key sk and an encapsulation c and outputs either the key K or an error $\perp$. For correctness, we require that if $(c, K) \leftarrow Encap(pk)$ then $K \leftarrow Decap(sk, c)$.

Note that unlike PKE, *Encap* has no message input. Instead, it internally generates a key.

IND-CCA Security for KEMs The adversary needs to distinguish between an encapsulated key K_0 (for which it gets the encapsulation c) and a randomly generated key K_1. See Figure 18.

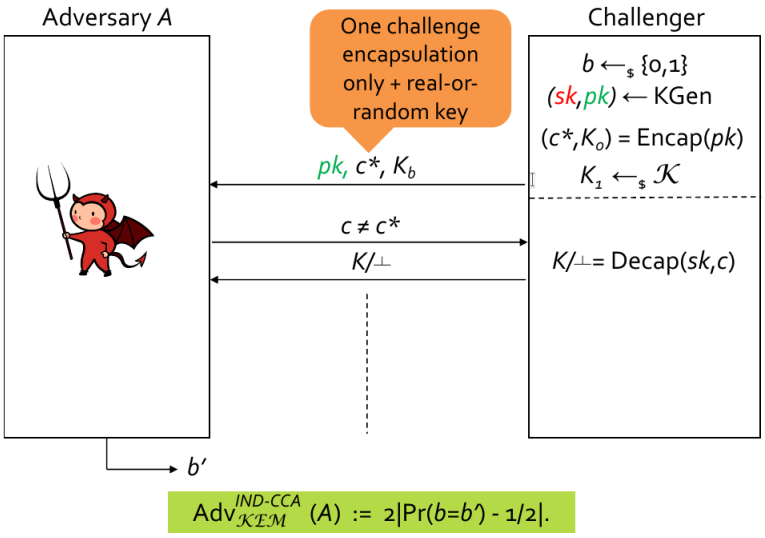

Figure 18: IND-CCA game for KEM

Data Encapsulation Mechanism DEM is simply a symmetric encryption scheme.

KEM/DEM Composition Paradigm We can build a PKE scheme from a KEM and a DEM:
$PKE.KGen$: $(sk, pk) = KEM.KGen()$; return (sk, pk)
$PKE.Enc$: $(c_0, K) = KEM.Encap(pk)$; $c_1 = DEM.Enc(K, m)$; return (c_0, c_1)
$PKE.Dec$: $m = \perp$; $K = KEM.Decap(sk, c_0)$; if $K \neq \perp$ then $m = DEM.Dec(K, c_1)$; return m

Theorem: If both KEM and DEM are IND-CCA secure, then so is the composed PKE. The proof proceeds using game hopping.

2.2 RSA

Textbook RSA
$KGen$: Generates random primes p, q of size $\frac{k}{2}$ bits. Sets $N = pq$. Also generates integers d, e such that $de = 1 \mod (p-1)(q-1)$. Outputs keypair (sk, pk) s.t. $sk = d$ and $pk = (e, N)$.
Enc: Outputs $c = m^e \mod N$.
Dec: Outputs $m = c^d \mod N$.

NOT secure. Not randomised, so cannot satisfy IND-CPA. Also malleable: $(m_0 \cdot m_1)^e = m_0^e \cdot m_1^e$.

Questions: How to generate p, q, d, e? How to encode messages as integers in $[1, N-1]$?

Generating keys Needs a good source of randomness and primality test. Things can still go wrong: shared prime factors[13], over-optimised prime generation, bad primality tests.

From e, d can be calculated using the Extended Euclidean algorithm (knowing p, q). In practice, $e = 2^{16} + 1 = 65537$ is often used for faster encryption (prime + likely coprime to $(p-1)(q-1)$).

Other optimisations lead to vulnerabilities:

- Small e: If e is small (e.g. $e = 3$), then for small messages it can happen that $c = m^3$ holds over the integers, i.e. without modular reduction. Then an attacker can simply take the (cube) root to recover m.
- Small d: insecure up for $d \leq N^{1/4}$ (Weiner's attack)

Keysize requirements By breaking the *Integer Factorisation Problem (IFP)* (to factor N) we can break RSA (the reverse may not hold!). One algorithm for the IFP is the *Number Field Sieve (NFS)*. Generally, the best known algorithms are super-polynomial, but sub-exponential.

See keylength.com for key size recommendations.

Malleability Since textbook RSA is malleable, an attacker can choose an arbitrary s and amend the message: $s^e \cdot c \mod N$ decrypts to $s \cdot m \mod N$.

Padding Goals: introduce randomness, expand short messages to full size, and destroy algebraic relationship (removing malleability property) to achieve IND-CCA security.

PKCS#1 v1.5 Padding Structure: Two most significant bytes set to `0x00 0x02`, then ≥ 8 random non-zero bytes, then a zero byte and finally the message m as the left most bytes. Implies a maximum message size of $k - 11$ bytes (assuming N as k bytes).

Encryption: $pad(m)^e \mod N$
Decryption: $m' = c^d \mod N$, then check for correct padding and return m.

Not IND-CCA secure.

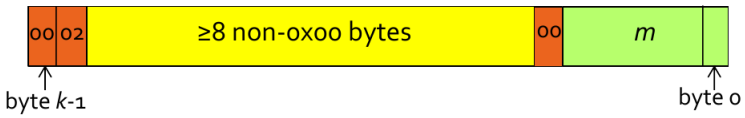

Figure 19: PKCS#1 v1.5 Padding

Bleichenbacher's Attack on PKCS#1 v1.5 Let $m' = (00\ 02 \mid\mid r \mid\mid 00 \mid\mid m)$ be the encoded message and let $c = m'^e \mod N$. Attacker asks oracle to decrypt $s^e \cdot c \mod N$. With probability $\approx 2^{-16}$ the padding is valid and decryption does not return a padding error.
Through an *adaptive attack*, using carefully chosen s, one can eventually recover m' (and thus m).

Originally this required around 2^{20} queries but has since improved to 5-10K queries for a 1024 bit modulus. For attacks against TLS, see DROWN and ROBOT.

[13] See "Mining your p's and q's."

RSA-OAEP Padding Optimal Asymmetric Encryption Padding, standardised in PKCS#1 v2.1, yet not as widely adopted.

Setup: Let n be the bitsize of N. Let k_0, k_1 be such that no adversary can perform neither 2^{k_0} nor 2^{k_1} operations in reasonable time (e.g. 128). Messages are assumed to be bitstrings of length $n - k_0 - k_1$. Let G, H be hash functions such that $G : \{0,1\}^{k_0} \mapsto \{0,1\}^{n-k_0}$ and $H : \{0,1\}^{n-k_0} \mapsto \{0,1\}^{k_0}$.

Encoding: See Figure 20. At the end, compute $c = (X \mid\mid Y)^e \mod N$.

Decoding: decrypt, then reverse the encoding, then check for the presence of the zero bits.

Can be proven to be IND-CCA secure (under strong number theoretic assumptions, including the Random Oracle Model). Intuition: output of hash functions is random, breaking up any algebraic relationships.

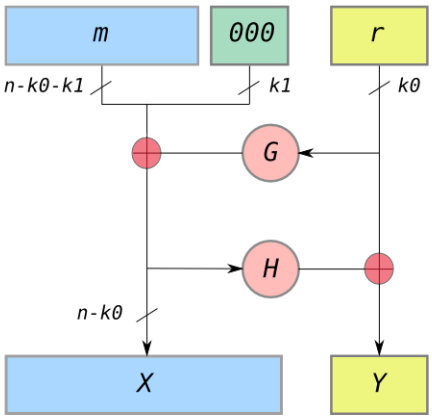

Figure 20: RSA-OAEP Padding

Random Oracle Model ROM Strong abstraction of hash functions: models hash functions as truly random functions.

A practical problem for proofs is that evaluating a random function can take exponential time – but we only consider poly-time adversaries! We solve this by "stopping time" when the adversary invokes a hash function, and forcing the adversary to use an oracle to evaluate the hash function for them. This extra oracle can inspect the adversary's queries, sample new random values for fresh queries, and return the values for repeated queries from its cache.

RSA (Inversion) Problem Summarizes the task of performing an RSA private-key operation given only the public key. Specifically for decryption, the task is to compute m given only c and (e, N).

One – but not the only – way to solve this is by factoring N. There is no proof that the IFP or the RSA problem are (computationally) hard. The RSA problem is at least as easy as the IFP, but it might be easier. (Finding d is equivalent to factoring though.)

2.3 Discrete Logarithm / Diffie-Hellman

Discrete Logarithm Setting Let p, q be large primes such that q divides $p - 1$, i.e. $p = kq + 1$. Now choose a random h and compute $g = h^{(p-1)/q} \mod p$ (repeat until $g \neq 1$). Observations:

- If $g \neq 1$ then the q powers of g – namely $G_q = \{g, g^2, g^3, \ldots, g^q\}$ – are all distinct mod p.

- $g^q = 1 \mod p$
- Multiplying any two elements of G_q results in another element of G_q.

Together, this means that G_q forms a *group* under multiplication mod p. In particular, G_q is a cyclic group with generator g and prime order q (number of group elements).

Discrete Logarithm Problem DLP Let G_q be a cyclic group with generator g and prime order q. Set $y = g^x \mod p$ where x is uniformly random in $\{0, 1, \ldots, q-1\}$[14]. Given (p, q, g) and y, find x.

Algorithms to solve the DLP include the *Function Field Sieve* (sub-exponential but super-polynomial in $\log p$), the *Pollard-ρ Algorithm* (exponential in $\log q$), and *baby-steps-giant-steps*.

Diffie-Hellman Key Exchange DHKE The group and (p, q, q) are public parameters. Alice picks a random x and sends $X = g^x \mod p$ to Bob. Bob picks a random y and sends $Y = g^y \mod p$ to Alice. Both compute a shared key $K = Y^x = X^y = g^{xy} \mod p$.

This is the modern view of the dynamic DHKE, where the private values x, y are considered *ephemeral* and are generated freshly each time. In the original paper the key exchange was static – participants would upload their key share to public, authentic directory.
Furthermore in applications we don't usually use K directly but use a KDF to derive keys from it.

Note that this is NOT yet public-key encryption!

MITM Attack on DHKE An active MITM attacker can trivially run two DHKEs with both parties and relay messages, thus completely compromising the security of the "modern" interactive DHKE. As a countermeasure, we need to provide authenticity of the public key shares. This can be done either using MACs (requires a pre-shared symmetric key – but DHKE still gives forward secrecy) or using signatures (requires a PKI).

Computational Diffie-Hellman Problem CDHP Given (p, q, q) and $g^a \mod p$ and $g^b \mod p$ find $g^{ab} \mod p$.

There is no proof that CDHP is equivalent to DLP (it could be easier), but is believed to be equivalent.

Decisional Diffie-Hellman Problem DDHP Given (p, q, q) and u.a.r. values a, b, c distinguish between (g^a, g^b, g^{ab}) (a *DDH triplet*) and (g^a, g^b, g^c).

It is believed that the DDH assumption is stronger than the discrete logarithm assumption, because there exist cyclic groups where the DLP is hard but the DDHP is easy.

[14]Note that $g^0 = 1 = g^q$.

ElGamal PKE scheme Public group parameters (p, q, g).
KGen: Choose $x \leftarrow \$\{0, 1, \ldots, q-1\}$ u.a.r.[15]. Set $sk = x, pk = X = g^x \mod p$.
Enc: Choose $r \leftarrow \$\{0, 1, \ldots, q-1\}$ u.a.r. and set $Y = g^r \mod p$. Compute $Z = X^r \mod p$. Compute $C' = M \cdot Z \mod p$. Output ciphertext $C = (Y, C')$.
Dec: Check that $Y \in G_q$ (else return $\perp$)[16]. Compute $Z' = Y^x \mod p$. Output $M = C' \cdot (Z')^{-1} \mod p$.
Correctness: left as an exercise.

Intuition: combine a long-term and a one-time key share. Use the resulting DH value Z as an encryption mask.

IND-CPA secure under the DDH assumption.
NOT IND-CCA secure (find an attack!).

Diffie-Hellman Integrated Encryption Scheme DHIES
Motivation: Addresses ElGamal's problems: missing IND-CCA and having to encode messages as group elements.

Definition: Public parameters (p, q, g) and a hash function H with a suitable output domain.
KGen: Choose $x \leftarrow \$\{0, 1, \ldots, q-1\}$ u.a.r.. Set $sk = x, pk = X = g^x \mod p$.
Enc: Choose $r \leftarrow \$\{0, 1, \ldots, q-1\}$ u.a.r. and set $Y = g^r \mod p$. Compute $Z = X^r \mod p$. Set $K = H(Z, X, Y) = K_e \mid\mid K_m$. Compute $C' = EtM(M)$ using K_e, K_m as encryption and MAC keys. Output $C = (Y, C')$.
Dec: Check that $Y \in G_q$ (else return $\perp$). Compute $Z' = Y^x \mod p$. Recompute K and decrypt.

Intuition: effectively a KEM/DEM construction. Can plug in any AE scheme.

IND-CCA secure in the Random Oracle Model.

2.4 Signatures

Signature scheme Goal: provide integrity. Public key analogue of MAC schemes.

KGen: generates a random key pair (sk, vk) = (signing key, verification key)
Sign: takes an input a signing key sk and a message m and outputs the signature σ
Vfy: takes as an input (vk, m, σ) and outputs 0 or 1[17].
Correctness: for all (sk, pk) produced by *KGen*[18] and for all $m \in \{0,1\}^*$ if $\sigma = Sign(sk, m)$ then $Vfy(vk, m, \sigma) = 1$.

UF-CMA Security We define *(Existential) Unforgeability under Chosen Message Attack (E)UF-CMA* through the security game given in Figure 21. We give the adversary access to vk and a signing oracle.

Informally: it should be hard for an adversary to produce a valid signature for a fresh message m^*.
Formally: The adversary wins if m^* is distinct from all queried m and $Vfy(vk, m^*, \sigma^*) = 1$.

Implication: if vk is bound to an identity, one cannot deny having signed m (*non-repudiation*)[19].

[15] $\leftarrow \$$ denotes drawing at random.
[16] There are attacks if $Y \notin G_q$.
[17] Some schemes perform "message recovery": they recover m from σ rather than receiving m as input.
[18] Notice that this definition says nothing about maliciously generated key pairs.
[19] Can MACs offer this?

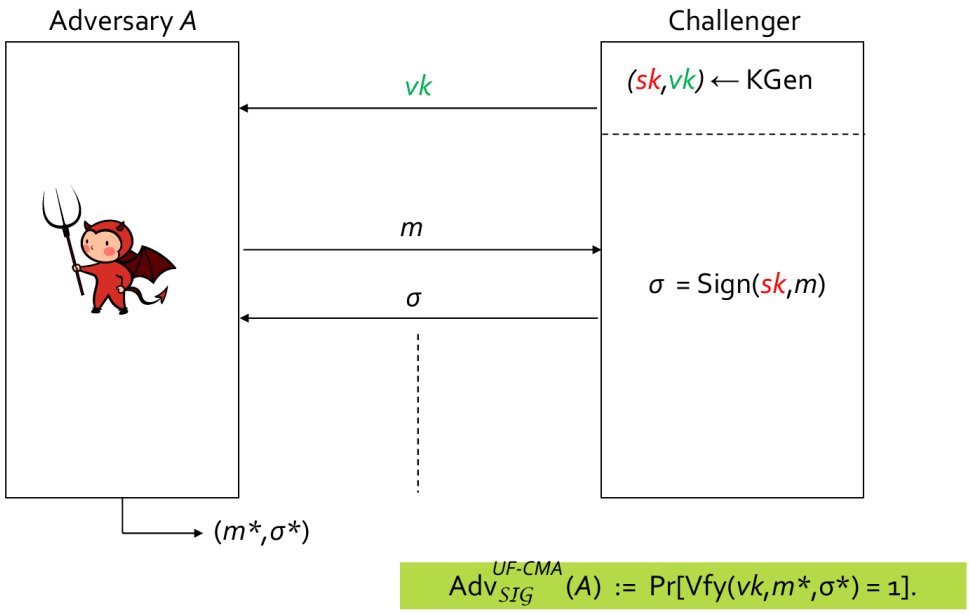

Figure 21: UF-CMA game

SUF-CMA Security Strong unforgeability: same game as UF-CMA.

Informally: it should be hard for an adversary to produce a fresh valid signature for a some message m^*. Formally: The adversary wins if (m^*, σ^*) is distinct from all queried pairs (m, σ) and $Vfy(vk, m^*, \sigma^*) = 1$.

We give the adversary more power: they can also win by producing a new signature on a previously queried message. Note that SUF-CMA and UF-CMA are equivalent for schemes with unique signatures.

Digital Signature Algorithm DSA DLP-based, public parameters (p, q, g).

$KGen$: Choose random $sk = x$ and compute $vk = X = g^x \mod p$.
$Sign$:

1. Generate a random $k \in \{1, ..., q-1\}$
2. Compute $r = (g^k \mod p) \mod q$
3. Compute $k^{-1} \mod q$ and hash $H(m)$
4. Compute $s = k^{-1} \cdot (H(m) + x \cdot r) \mod q$
5. Output $\sigma = (r, s)$

Vfy:

1. Check that $r, s \in \{1, ..., q-1\}$
2. Compute $w = s^{-1} \mod q$
3. Compute $u_1 = w \cdot H(m) \mod q$ and $u_2 = w \cdot r \mod q$
4. Output 1 if $(g^{u_1} X^{u_2} \mod p) \mod q = r$ else 0

Correctness: left as an exercise.

DSA Security Informally: A hash collision results in a forgery. Usual DLP attack on sk.
Formally: No good security proof for DSA is known.

Catastrophic failure under randomness failure:
Assume the same k is used to sign two different messages. This is detectable since then $r_1 = r_2$. This knowledge allows to rearrange the two formulas, recover k and solve for $sk = x$!
This issue is real: OpenSSL 2008, PlayStation 2010, Android 2013, any virtualized environment.
We can *hedge* against this by *derandomising*: generate pseudo-random $k = F_K(vk||m)$. Need to keep the PRF key K together with sk.

Naïve RSA signature Sign using $\sigma = m^d \mod N$. Verify by $\sigma^e \stackrel{?}{=} m \mod N$.

INSECURE. Trivial forgery: $\sigma_1\sigma_2 \mod N$ is a signature on m_1m_2.

Full-Domain Hash (FDH) RSA signature Sign using $\sigma = H(m)^d \mod N$. Verify by $\sigma^e \stackrel{?}{=} H(m) \mod N$.

Hash function destroys algebraic (multiplicative) structure. Allows signing long messages. But: needs a hash function with the output domain $\{0, ..., N-1\}$.

Theorem: If the RSA problem is hard and H is a random oracle then FDH is UF-CMA secure.

Hash-based RSA signature with padding Sign using $\sigma = pad(H(m))^d \mod N$ for some padding function pad. Verify by $\sigma^e \stackrel{?}{=} pad(H(m)) \mod N$.

Relaxes requirement of hash function having a matching output domain. Used e.g. in PKCS#1 v1.5 with padding 00 01 FF ... FF || c || $H(m)$ for constant c.
Bleichenbacher applies (if padding oracle in implementation). No known security proof for scheme with PKCS#1 v1.5 padding.

RSA Probabilistic Signature Scheme RSA-PSS See Figure 22 where H, G_1, G_2 are hash functions with suitable output lengths.
Sign using $\sigma = (0||s||t||u)^d \mod N$. Verify by $(b||s'||t'||u') = \sigma^e \mod N$, computing $r' = G_1(s') \oplus t'$ and then re-encoding from the start and checking for a match.

Standardised e.g. in PKCS#1 v2.1.

Theorem: If H, G_1, G_2 are random oracles then the UF-CMA security of RSA-PSS is tightly related to the hardness of the RSA problem.

2.5 Elliptic Curves

Motivation: key sizes Cryptographic algorithms provide different security levels (measured as entropy in bits (?)) depending on their design and the key size (RSA: modulus size, ECC: group size, DL: field+subgroup sizes).
We can define DL-based algorithms over any cyclic group, including elliptic curves. Since there are no sub-exponential algorithms known in general, we get away with smaller key sizes than e.g. RSA, leading to performance benefits.

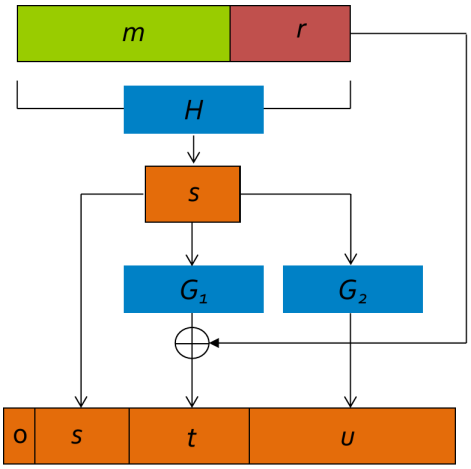

Figure 22: RSA-PSS

Basics An elliptic curve E over a field F is a set of pairs $(x, y) \in F \times F$. There are multiple ways of representing the curves, e.g. the Montgomery form, the Edwards form, or the short Weierstrass form using affine coordinates:

$$E = \{(x, y) \in F \times F | y^2 = x^3 + ax + b\} \cup \{O\}$$

where $4a^3 + 27b^2 \neq 0$ over F and O is the special "point at infinity".

Addition: Two points can be added to obtain a third point on the curve. O acts as an additive identity: $P + O = O + P = P$. Each point $P = (x, y)$ as an additive inverse $-P = (x, -y)$ such that $P + (-P) = O$ (and O is its own inverse).
Visually, when $P + Q = S$ then R is the unique point where the straight line $\overline{PQ}$ intersects E and S is the projection of R through the x-axis. For the detailed formula, see the slides.

Addition turns the set of curve points into a group (with group operation "+" and group order equal to the number of points on the curve). In practice, we work in a prime-order subgroup to maximise security against generic attacks.

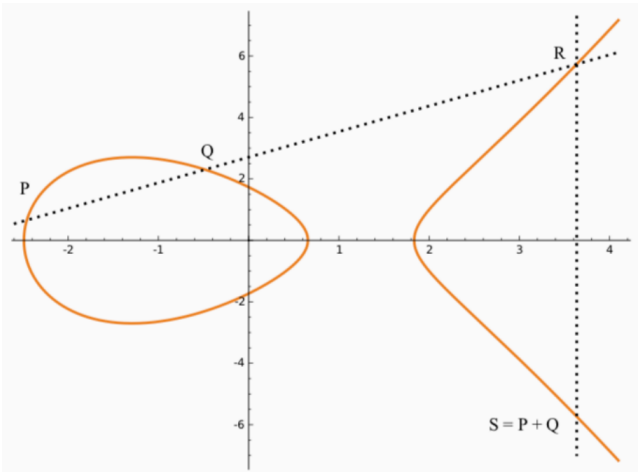

Figure 23: Elliptic Curve example

Scalar multiplication by k: We write $[k]P$ for the operation of adding P to itself k times. Consider the following analogue: $[k]P$ on $E \leftrightarrow g^x \mod p$. Note that in general $[k]P \neq (kx, ky)$! Scalar multiplication can be implemented using double-and-add.

Point compression: We can exploit the symmetry of the curve to avoid having to store two field elements per point: we can store only the x-coordinate and the "sign" of y. Instead of $2 \cdot \log_2 p$ bits this only needs $1 + \log_2 p$ bits.

Elliptic Curve Discrete Logarithm Problem ECDLP Let E be an elliptic curve over the field F of prime order p. Let P be a point of prime order q on E. Set $Q = [x]P$ where x is u.a.r. from $\{0, 1, ..., q-1\}$. Given E, P, Q, find x.

The best known algorithms for solving the ECDLP are the generic ones that work for any cyclic group and running in time $\mathcal{O}(q^{1/2})$.

Curve selection can be tricky and needs careful consideration (Which field F? Which curve E? Which base point P of large prime order q?). It is safest to rely on standardised curves, e.g. NIST P-256 or Curve25519.

Elliptic Curve Diffie-Hellman Ephemeral ECDHE Similar to DHKE. Alice chooses x and sends $[x]P$, Bob chooses y and sends $[y]P$, then both share the secret $[x][y]P$.

Key pair generation Choose k u.a.r. from $\{0, 1, ..., q-1\}$. Then k is the private key and $Q = [k]P$ the public key.

Elliptic Curve Integrated Encryption Scheme ECIES Similar to DHIES. See the slides for details.

Elliptic Curve Digial Signature Algorithm ECDSA Similar to DSA. See the slides for details. Also suffers catastrophic failure under nonce reuse.
Malleable: if (r, s) is a valid signature on m under vk then so is $(r, -s)$. Thus cannot be SUF-CMA secure, but is plausibly UF-CMA secure.

ECC Adoption Originally slow takeoff due to mathematical complexity, lack of mature standards, unclear patent situation, general FUD. Introduced in the 1980s, only took of in the 2010s.

2.6 Key Management

Motivation Cryptography shifts the problem of security data to the problem of managing keys.

Aspects key generation, key storage, key distribution, key usage + scope, key replacement/rollover, key backup + recovery, key revocation + destruction.

Key lifecycle Pre-operational → operational → post-operational[20] → destroyed

Principle of Key Separation A cryptographic key should only be used for its intended purpose.

Any given key should not be used in more than one cryptographic algorithm. Reasoning: the interaction between algorithms may lead to unexpected weaknesses. E.g.: CBC mode encryption and CBC-MAC, or RSA encryption and signatures.

[20] Don't use the key any more but keep it around to e.g. decrypt past communications again.

3 Advanced Cryptography

3.1 Data at rest

3.1.1 Searchable Encryption

Searchable Encryption consists of three protocols:

1. Setup: Client generates an *encrypted database + encrypted search index*[21], uploads them to the server.
2. Search: Client generates a *search token*, server uses it to process the search, returns the result.
3. Update: Client generates an *update token*, server uses it to update the encrypted database and encrypted search index, returns success/failure.

Active field of research, many open problems (e.g. leakage analysis + prevention).

Goals

- Security: Confidentiality (of data and queries) against an *honest-but-curious* server.[22]
- Efficiency: Storage, computational, bandwidth requirements.
- Functionality: Supported query types.

First construction Idea: randomly choose a PRF F_K and replace each keyword w in the index by $F_K(w)$. Also encrypt each document under some symmetric key K_0.

Leakage from setup $\mathcal{L}_{Setup}$: total number of documents and keywords, keyword frequency, co-occurences of keywords.

Leakage from searches $\mathcal{L}_{Search}$: result patterns, query patterns, result intersection between queries.

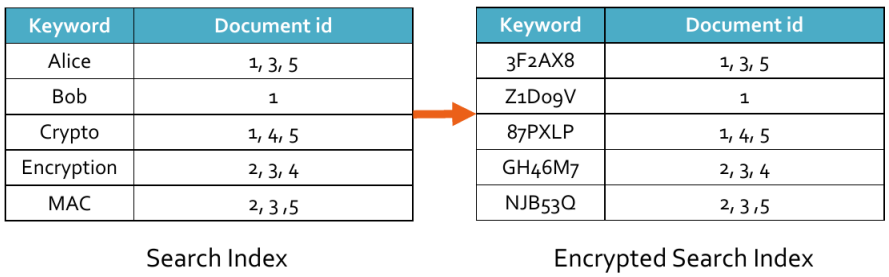

Figure 24: First construction for searchable encryption

Second construction Idea: randomly choose a PRF F_K. For each keyword w calculate $K_1||K_2 = F_K(w)$. Replace each keyword with K_1 (as before). In addition: associate each document id for w with a counter cnt and replace the id with $id \oplus F_{K_2}(cnt)$.

This hides the co-occurences of keyword pairs (PRF security).

[21] A search index is a mapping from document ids to keywords, and/or vice versa. For more details see the lecture *Information Retrieval*.

[22] Compare this security model against a fully malicious server.

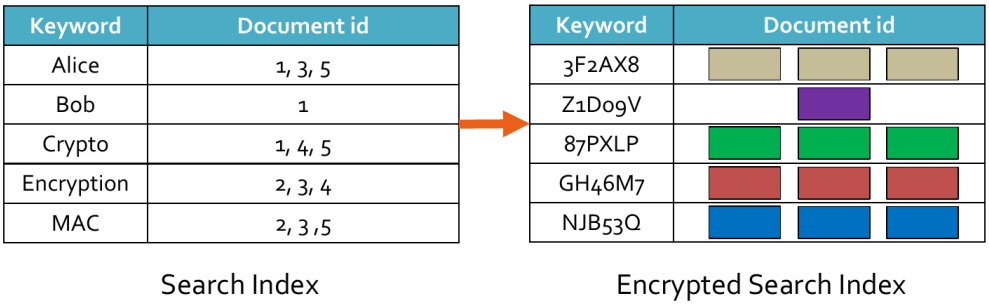

Figure 25: Second construction for searchable encryption

Third construction Idea: ... (as before) ...
In addition: associate each document id for w with a counter cnt and replace the id with $id \oplus F_{K_2}(cnt)$. Then store this document id value in a key-value store (dictionary) under "key" $F_{K_1}(cnt)$.

Search: Send $(K_1, K_2, |cnt_{querystring}|)$ to the server. Server recomputes the $F_{K_1}(cnt)$ to find the values in the dictionary. Then decrypts them to document ids using K_2. Looks up the encrypted documents and returns them.

Leakage from setup $\mathcal{L}_{Setup}$:[23] total number of documents.

Leakage from searches $\mathcal{L}_{Search}$: (same as before) result patterns, query patterns, result intersection between queries.

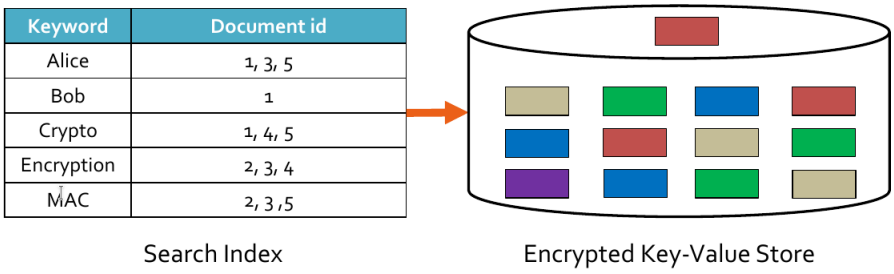

Figure 26: Third construction for searchable encryption (setup)

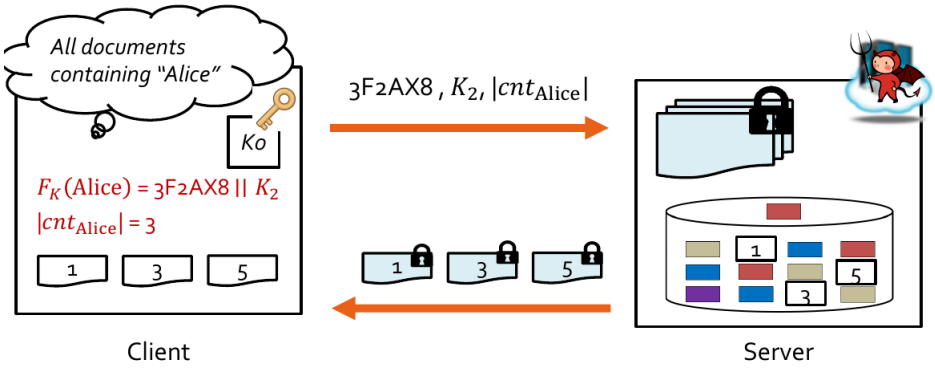

Figure 27: Third construction for searchable encryption (search)

Defining Leakage Goal: formally define the leakage $\mathcal{L} = (\mathcal{L}_{Setup}, \mathcal{L}_{Search})$ of searchable encryption schemes.

[23] This is also what an adversary learns from a single snapshot.

Security game: an adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ that contains either the real world or a simulator. The simulator $\mathcal{S}$ only has access to the leakage $\mathcal{L}$ (but not to the secret key). Intuitively, the adversary should gain no extra information other than the leakage.

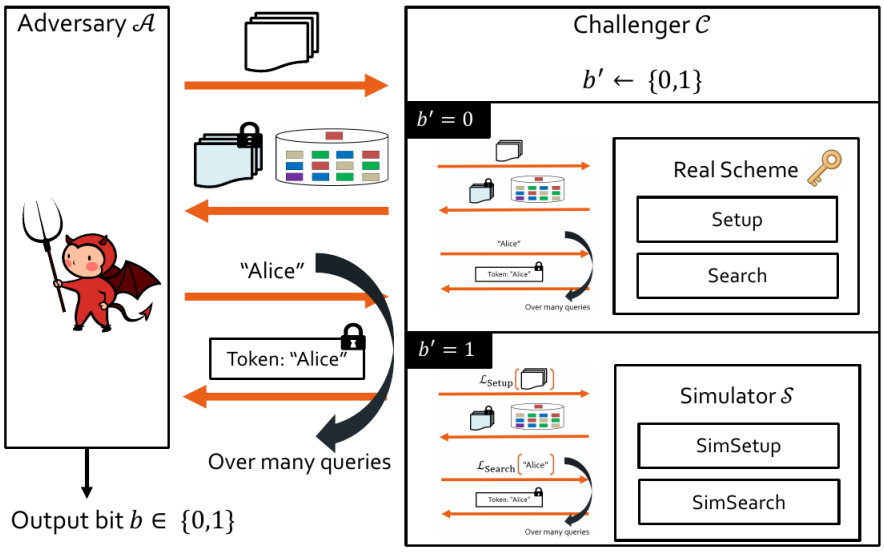

Figure 28: Searchable encryption leakage game

Analysing Leakage What can the adversary infer from the leakage? It turns out that given auxiliary information, one can e.g. perform query recovery.

3.2 Data in transit

3.2.1 Transport Layer Security TLS

Goal "Secure channel between two peers" – confidentiality, authentication, integrity in the face of a network adversary and only requiring a reliable in-order transport mechanism.

Architecture

- Handshake protocol: Cipher suite negotiation, peer authentication, key material establishment
- Record protocol: "normal" data transfer
- Many other features: session resumption, renegotiation, extensions, ...

For the format of the handshake protocol and a TLS record, see Figure 30 and Figure 31. These blueprints are instantiated using the negotiated cipher suite: for example RSA for the key exchange and authentication with AES128-CBC as a cipher and SHA1 as a MAC.

Ad handshake: the peers create/derive a pre-master secret during the handshake. Authentication is done via a signature over the entire handshake transcript (`ServerCertificateVerify`). MITM protection via MAC over transcript (`ServerFinished`). This is an example of *Sign-and-Mac*.

Security Issues Various, due to protocol complexity. Protocol issues, padding oracles, implementation bugs, downgrade attacks.

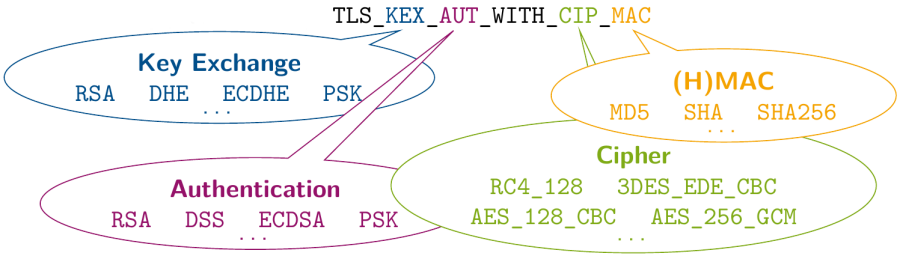

Figure 29: TLS Cipher Suite Format ($\leq$ TLS 1.2)

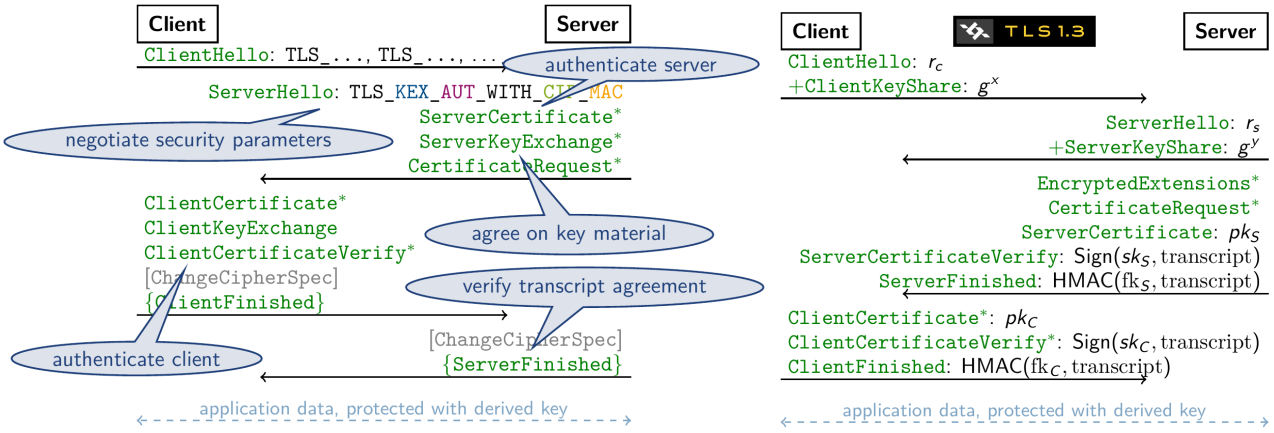

Figure 30: TLS Handshake Protocol (simplified, left: $\leq$ TLS 1.2 right: TLS 1.3)

TLS 1.3 Goals

- *Clean up:* remove legacy crypto, unused/broken features, static RSA/DH[24]
- *Improved latency:* TLS 1.2 handshake takes 2 round trips, "wasting" one to learn server capabilities. Instead, run full hand-shake in 1 round trip (possible due to feature reduction). 0-RTT handshake using a shared resumption secret PSK – at the sacrifice of FS and danger of replay attack.
- *Improved privacy:* Encrypt all most all handshake messages. Send a `[Client|Server]KeyShare` with the `[Client|Server]Hello`. Then encrypt all subsequent messages with the *handshake traffic key* tk_{hs}.
- *Continuity:* Interoperability (make `ClientHello` look like in TLS 1.2)
- *Security assurance:* Formally analyse the protocol before it is deployed (rather than after). This is done in various ways: computational model, tool-based, verified implementations.

Key Separation Detailed key schedule for deriving separate keys everywhere. Advantage: if a *key* is compromised, all other keys still remain secure (as long as the *secrets* from which they were derived are private).
In TLS 1.3: use $HKDF.Extract(salt, keymaterial)$ to extract entropy (e.g. from PSK) into a random key which can be expanded into longer random output using $HKDF.Expand(key, context, length)$. E.g. master secret MS, resumption master secret RMS, exporter master secret EMS, application data traffic key tk_{app}, ...

[24] They don't provide forward secrecy.

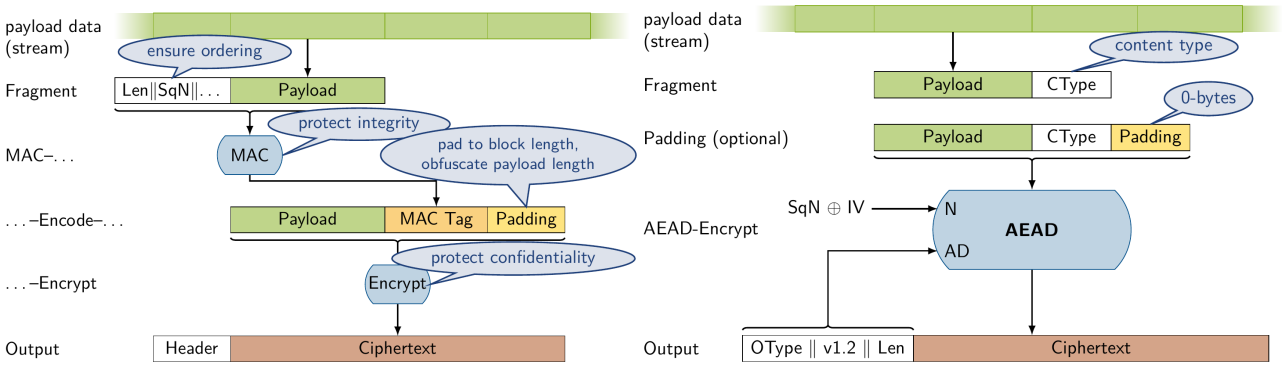

Figure 31: TLS Record Format (simplified, left: $\leq$ TLS 1.2 right: TLS 1.3)

3.2.2 Signal Messaging Protocol

Goals

- *Asynchronous:* Parties need not be online at the same time to run a key exchange or send messages.
- *Threat model:* Adversary controls the network, can reveal derived message keys, can corrupt long term secrets, can compromise ephemeral secrets.
- *Perfect Forward Secrecy PFS:* Key indistinguishability holds for all messages sent before the compromise.
- *Post-Compromise Secrecy PCS:* Parties can recover security if the adversary becomes passive. I.e. after a period of passiveness, future messages are secret again, even given knowledge of the long term secrets.

Architecture

1. *Registration Phase:* clients register their Prekey Bundle with the server
2. *Initialisation Phase:* clients run a key exchange
3. *Asymmetric Ratchet Phase*
4. *Symmetric Ratchet Phase*

Symmetric Ratchet Starting from a *chain key* ck_{i-1} use a KDF to derive both a new chain key ck_i and a *message key* mk_i. Then if a ck_j is exposed, only subsequent messages are compromised (achieving PFS wrt. ck and thus mk).

Asymmetric Ratchet Do a half-DH key exchange with each message. Start with a *root key* rk_{i-1} and a *ratchet key* $g^{A_{i-2}}$ known to Bob. Use the DH secrets as key to a KDF to derive root and chain keys from.
Alice *ratchets* rk forward by generating a new g^{A_i} that Bob will receive when he next comes online.[25]

[25]Note that the subscripts i are across both A's and B's DH shares.

Double Ratchet Combine symmetric and asymmetric ratcheting: the former computes ck, mk and the latter rk, ck. Asymmetric ratcheting happens on pin-pong messages, symmetric ratcheting happens on successive messages.

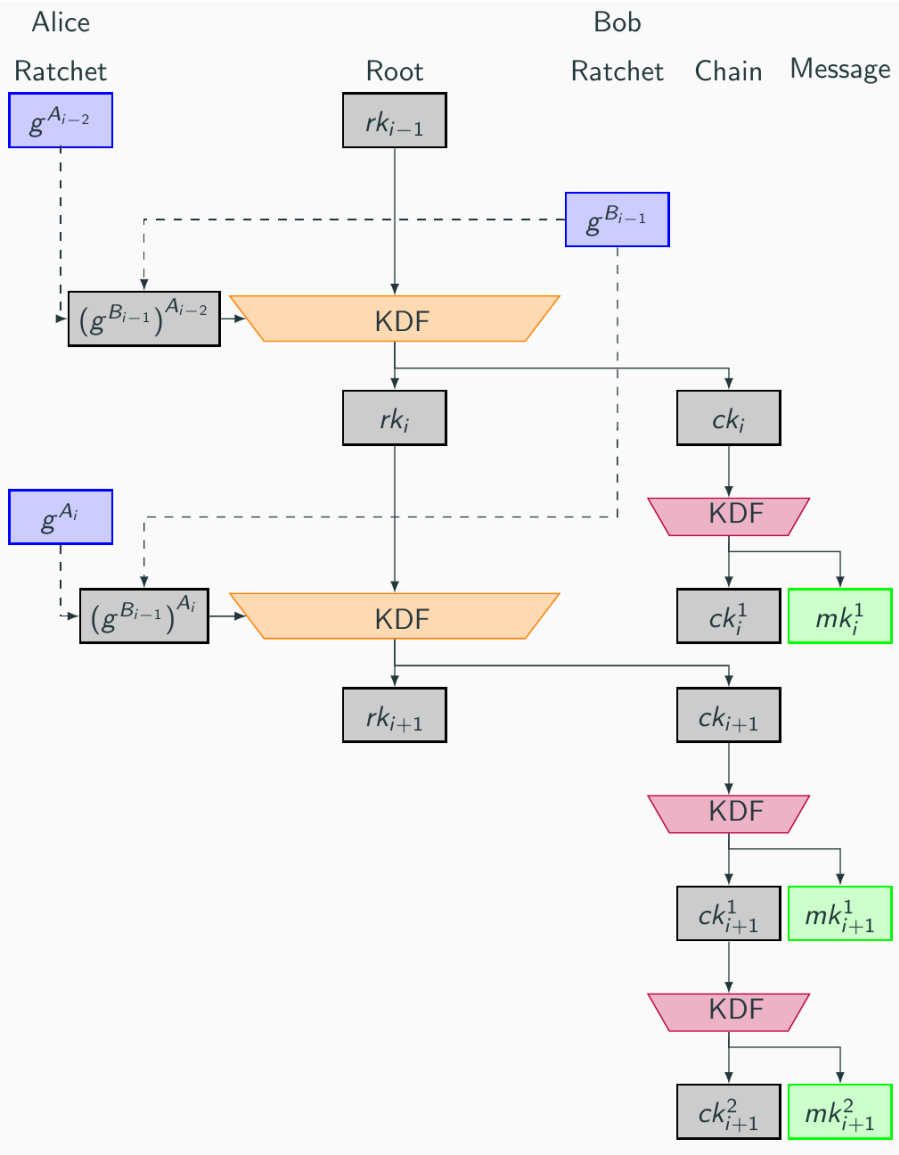

Figure 32: Signal Double Ratchet

Registration phase Alice generates a *PreKey Bundle* that she uploads to the server. It contains:

- id_A – her long term identifier (e.g. phone number)
- $(idk_A, idpk_A = g^{idk_A}) \leftarrow \$\ DHGen$ – Identity Key (long term)
- $(spk_A, sppk_A = g^{spk_A}) \leftarrow \$\ DHGen$ – Signed Prekey (medium term)
- $(otk_A, otpk_A = g^{otk_A}) \leftarrow \$\ DHGen$ – One-Time Key (short term)
- $\sigma_A = SIG.Sign_{idk_A}(sppk_A)$ – signature

Then $PKB_A = \{id_A, idpk_A, sppk_A, otpk_A, \sigma_A\}$.

There is no PKI binding identities to public keys. Thus *Unknown Key Share (UKS)* attacks on DH are possible. Users must manually verify fingerprints (Safety Numbers).

Initialisation Phase – X3DH Bob gets Alice's PKB_A from the server and generates a fresh one-time key $(otk_B, otpk_B = g^{otk_B})$ as well as a *ratchet key* $(rck_B^0, rcpk_B = g^{rck_B})$. All of these are combined in the *Extended Triple Diffie-Hellman Key Exchange (X3DH)* – in different pairwise combinations – to produce a symmetric *master secret* ms and the first root and chain keys rk^0, ck^0.
From there, we can run the normal Double Ratchet.

The combinations of DH key shares are carefully chosen:

- $pms_3 = otpk_A - otk_B$ gives forward secrecy
- $pms_2 = sppk_A - otk_B$ gives freshness if $otpk_A$ is missing, and prevents *Key Compromise Impersonation KCI*[26] attacks against Alice
- $pms_1 = idpk_A - otk_B$ prevents KCI against Bob, and provides Alice authentication
- $pms_0 = sppk_A - idk_B$ provides Bob authentication
- Lack of $idpk_A - idk_B$ provides *deniability* (especially if otk, spk_A are later leaked then anyone could generate a transcript)

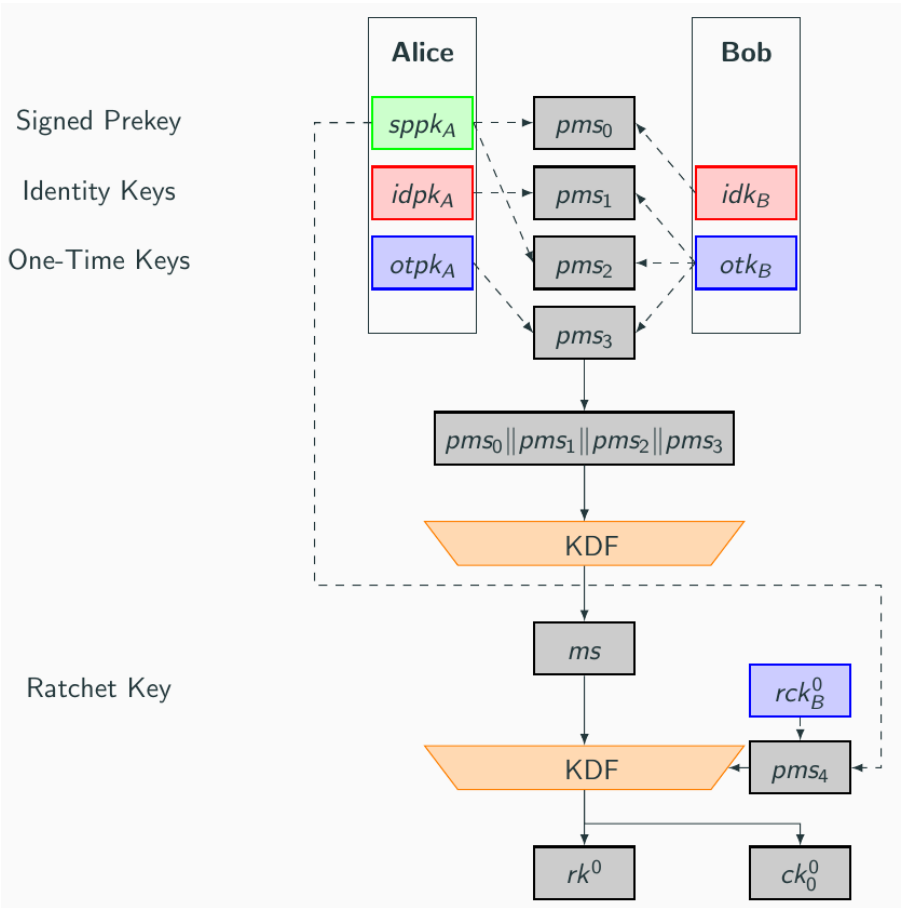

Figure 33: Signal X3DH

[26] An attack who learns your long term secret key can impersonate you.

Message encryption Uses AEAD encryption under mk. The associated data is $AD = rcpk_A^i||idpk_A||idpk_B||PN||ctr$. $rcpk_A^i$ is the most recently public ratchet key generated by the sender. PN is the number of messages in the last chain (allowing the receiver to delete old ck values). ctr is the number of messages in the current chain so far (allowing out-of-order delivery).

All primitives are instantiated using reasonable, modern primitives.

Other areas Group messaging, multi-device, authentication (Safety Numbers), Private Contact Discovery, Private Groups.

3.2.3 Messaging Layer Security Protocol

TODO

3.3 Data under computation

3.3.1 Fully Homomorphic Encryption FHE

Motivation Allow data to be processed while remaining encrypted. Applications: Secure Outsourcing, Private Set Intersection (PSI), Private Machine Learning As A Service.

Challenges

- Crypto: Underlying math, parameter selection, security analysis.
- Computation Paradigm: No if/else, no loops, no jumps – all would leak information about the encrypted data. Optimisations, approximations, SIMD batching.

(Ring-) Learning With Errors (R)LWE The LWE problem is conjectured to be hard (even on quantum computers). Simplified idea: mask data with random noise. Need to carefully define encryption/decryption, addition and multiplication. For multiplication, we need *relinearization* to make the maths work.

Problem: noise increases with each operation and will eventually grow too large for decryption to succeed. Solution: use *bootstrapping* to (homomorphically) reduce the noise again. Effectively, bootstrapping produces an encryption of the same encrypted message, but at a lower noise level.[27] But: bootstrapping is complex and slow (order of seconds to minutes), so we try to avoid it in practice.

Parameter selection Careful trade-off between efficiency, security and correctness. Security based on current knowledge on how hard the LWE problem is.

Tool support "Libraries" implement the underlying FHE schemes, addressing the crypto side. "Compilers" help with writing higher-level code, addressing the computation paradigm concerns. E.g. SEAL (C++), EVA (Python), nGraph-HE (Tensorflow), SEALion (Keras).

[27] This is one of the practical breakthroughs of Gentry '09.

Encryption: $let\ \boldsymbol{\mu} = \boldsymbol{m} * \left\lfloor \frac{q}{t} \right\rfloor for\ \boldsymbol{m} \in R_t, t \ll q$

$$\boldsymbol{c}_0 = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{\mu} + \boldsymbol{e}$$

$$\boldsymbol{c}_1 = \boldsymbol{a}$$

$$\boldsymbol{Enc_s(m)} \coloneqq (\boldsymbol{c}_0, \boldsymbol{c}_1)$$

Addition: $\boldsymbol{Add}(\boldsymbol{c}, \boldsymbol{c}') \coloneqq (\boldsymbol{c}_0 + \boldsymbol{c}'_0, \boldsymbol{c}_1 + \boldsymbol{c}'_1)$

Multiplication:

$$\boldsymbol{Mult}(\boldsymbol{c}, \boldsymbol{c}') \coloneqq \frac{t}{q}(\boldsymbol{c}_0 \cdot \boldsymbol{c}'_0, \boldsymbol{c}_0 \cdot \boldsymbol{c}'_1 + \boldsymbol{c}_1 \cdot \boldsymbol{c}'_0, \boldsymbol{c}_1 \cdot \boldsymbol{c}'_1)$$

Relinearization:

$$\boldsymbol{Relin}(\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2) \coloneqq (\tilde{\boldsymbol{c}}_0, \tilde{\boldsymbol{c}}_1)$$

Bootstrapping:

$$\boldsymbol{Bootstr}\ (\boldsymbol{Enc_s(m)}) \coloneqq \boldsymbol{Enc_{s'}(m)}$$

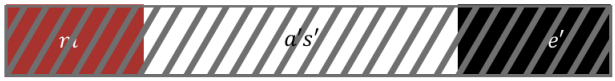

Figure 34: FHE: RLWE operations