Applied Cryptography
Spring Semester 2023
Lectures 14, 15, 16 and 17

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

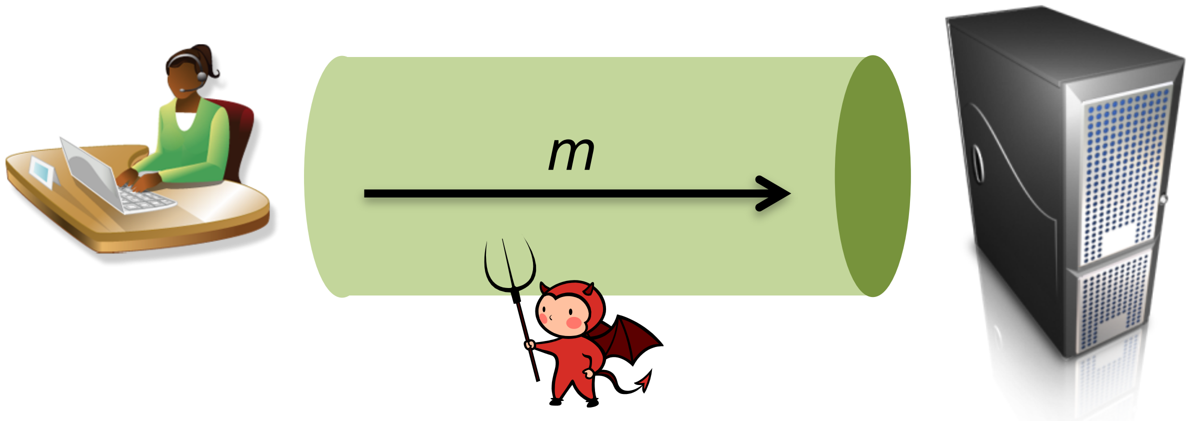
Overview of lectures 14, 15, 16, 17

- Introducing Message Authentication Codes (MACs)
- Security definition for MACs
- Constructions for MACs:
  - MACs from PRFs
  - Domain extension for a MAC using CR hash functions
  - HMAC
  - (CBC-MAC and CMAC)
  - MACs from universal hashing

Introducing MACs

Introducing Message Authentication Codes (MACs)

- Setting: assume Alice wants to send a message *m* to Bob
- Adversary controls the network link and wants Bob to accept a modified message *m'*.
- So Alice and Bob need a cryptographic primitive that detects message modifications.
- The primitive should provide integrity and data origin authentication of messages: guarantees that messages have not been tampered with in transit and that they originate from the expected sender
- In other words, no attacker should be able to forge messages.



Defining MACs

Definition:

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

$$\text{KGen}: \{\} \longrightarrow \{0,1\}^k$$

$$\text{Tag}: \{0,1\}^k \times \{0,1\}^* \longrightarrow \{0,1\}^t$$

$$\text{Vfy}: \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^t \longrightarrow \{0,1\}$$

For correctness, we require that for all K and all m:

if $\tau \leftarrow \text{Tag}(K,m)$, then $\text{Vfy}(K,m,\tau) = 1$.

Notes on MAC definition

- Keys K output by KGen are usually uniformly random k-bit strings.
- Typically, t, the tag length is small – perhaps 96 or 128 bits.
- We often write $\mathrm{Tag}_K(m)$ for $\mathrm{Tag}(K,m)$ and $\mathrm{Vfy}_K(m,\tau)$ for $\mathrm{Vfy}(K,m,\tau)$.
- Both Tag and Vfy can be randomised, but typically are deterministic.
- Correctness is a basic functionality requirement and **not** a security requirement.
- Boneh-Shoup refers to a signing algorithm S instead of a tagging algorithm Tag, and uses V for verification instead of Vfy.

Deterministic MACs

- When Tag is deterministic, we can always define a Vfy algorithm as follows:

  Given input (K,m,τ), Vfy runs Tag on input (K,m) to obtain τ'; output 1 if $\tau' = \tau$, and 0 otherwise.

- Such schemes have unique tags*:* given (K,m), there is a unique τ such that $\text{Vfy}(K,m,\tau)$ outputs 1, namely $\tau = \text{Tag}(K,m)$.

Security of MACs

Security for MACs

<u>MAC unforgeability (informal):</u>

It should be hard for an adversary who does not know the key K to forge a valid tag on a message, that is, to compute a pair (m', τ') for which $\mathrm{Vfy}(K, m', \tau') = 1$.

- Even if the adversary has seen many message/tag pairs (m, τ) computed using the Tag algorithm with the same key K.
- Even if the adversary gets to choose all the messages m – chosen message. attack with tag oracle.
- Extended definition: And even if the adversary can learn whether its attempts to forge MAC/tag pairs are correct or not – verify oracle.

Security for MACs

- The security definition means that the MAC tag τ must depend on every bit of the message *m.*

  Insecure example: Tag(K,m) = H($m_0 m_1 \ldots m_{k-1} \oplus K$)

- The security definition also means that it must be hard to recover K given some pairs (m,τ) (otherwise attacker can recover K, and then forgery is trivial).

  Insecure example: Tag(K,m) = $H(m) \oplus K$ where H is a hash function with output size k bits.

- We need to be careful about *length extension* attacks.

  Insecure example: Tag(K,m) = $H(K \,||\, m)$ where H is a Merkle-Damgård hash function.

  Insecure because, given Tag(K,m) = $H(K \,||\, m)$, we can compute

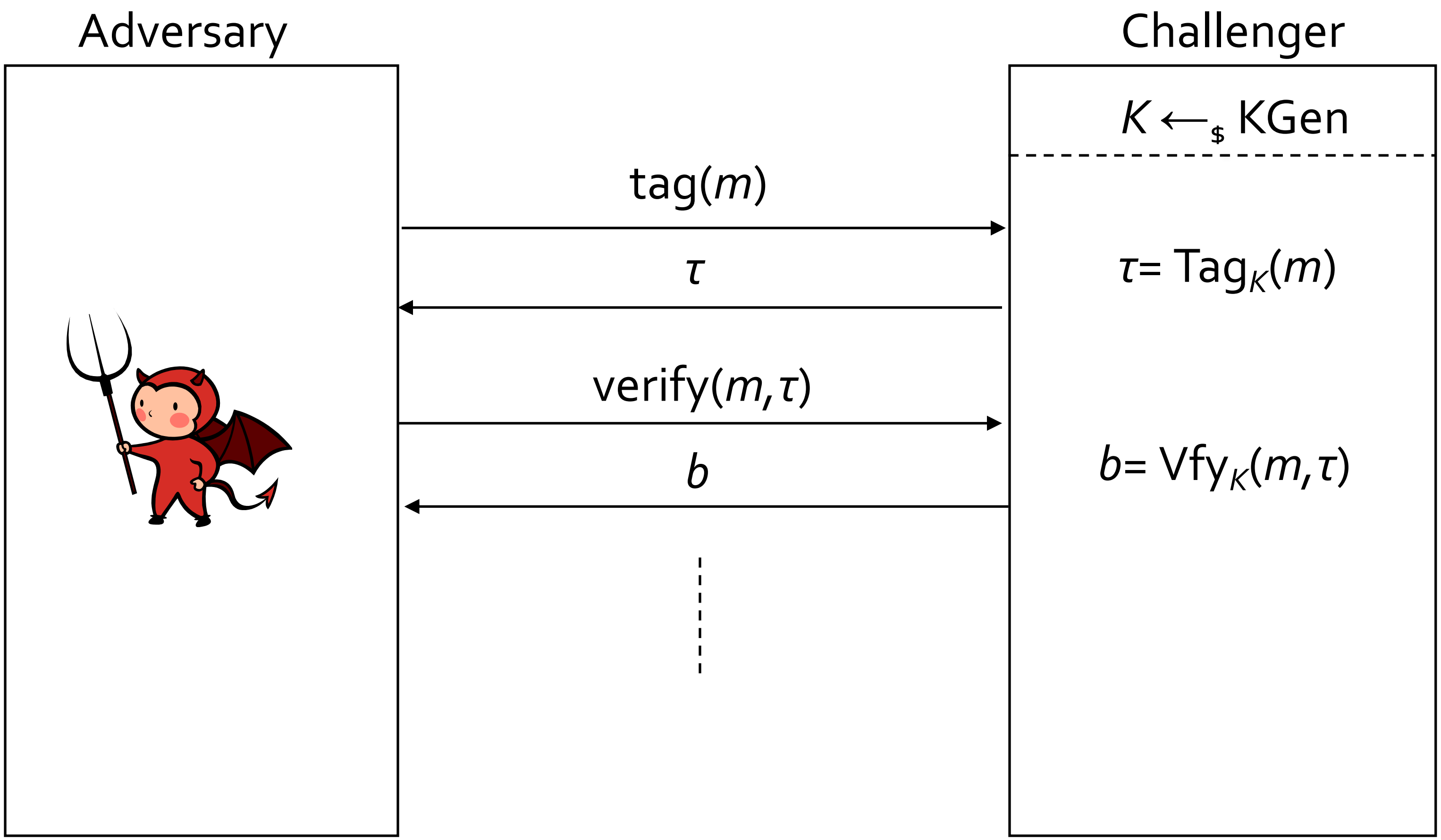
  $$\mathrm{H}(\mathrm{pad}(K \,||\, m) \,||\, m'')$$

  for any string m'' by exploiting length extension.

Formalising Security for MACs

- Security of a MAC scheme (KGen, Tag, Vfy) is formalised as a game between an adversary and a challenger.
- The challenger begins by running KGen to obtain a key K.
- The adversary has access to a tag oracle and a verify oracle.
  - The tag oracle accepts messages as input and returns corresponding tag values: $\tau \leftarrow \mathrm{Tag}_K(m)$.
  - Using the tag oracle, the adversary can obtain MAC tags for messages of its choice: a chosen message attack.
  - The verify oracle accepts pairs (m,τ) and returns $\mathrm{Vfy}(K,m,\tau)$.
  - Using the verify oracle, the adversary can test whether a given message/tag pair is valid or not.
- After accessing the tag and verify oracles as many times as it likes the adversary stops, and its success is determined.

MAC security game in a picture



Determining when the MAC adversary wins

- There are two flavours of unforgeability for MACs: weak unforgeability and strong unforgeability.
- In weak unforgeability, the adversary wins if verify(m^*,τ^*) = 1 for some query (m^*,τ^*) made to the verify oracle and no query tag(m^*) was made.
- In strong unforgeability, the adversary wins if verify(m^*,τ^*) = 1 for some query (m^*,τ^*) made to the verify oracle and no query tag(m^*) with response τ^* was made.
  - So in the strong case, some pair (m^*,τ^*) in a verify oracle query is **distinct** from all the pairs (m,τ) arising in tag oracle queries – but m^* could be repeated from a tag oracle query.
  - Such a "repeated m^*" would **not** be a win for a weak adversary.
  - So it is potentially easier for the adversary to win in the strong version.
  - Hence achieving strong unforgeability is harder than achieving weak unforgeability.
  - So strong unforgeability is a **stronger** security notion than weak unforgeability!

WUF-CMA security game in a picture

Adversary		Challenger
		$K \leftarrow_{\$}$ Kgen $S \leftarrow \{\}$
	tag(m) $\rightarrow$	
	$\leftarrow$ τ	$\tau \leftarrow \mathrm{Tag}_K(m)$ $S \leftarrow S \cup \{m\}$
	verify(m,τ) $\rightarrow$	
	$\leftarrow$ b	$b = \mathrm{Vfy}_K(m,\tau)$
	⋮	

A wins if it makes a query verify(m^*,τ^*) with output b=1 for some $m^* \notin S$.

SUF-CMA security game in a picture

Adversary | Challenger

$K \leftarrow_{\$} \mathrm{Kgen}$

$S \leftarrow \{\}$

tag(m) →

← τ

$\tau \leftarrow \mathrm{Tag}_K(m)$

$S \leftarrow S \cup \{(m,\tau)\}$

verify(m,τ) →

← b

$b = \mathrm{Vfy}_K(m,\tau)$

⋮

A wins if it makes a query verify(m^*,τ^*) with output b=1 for some $(m^*,\tau^*) \notin S$.

Formalising Security for MACs

- We say that a MAC scheme (KGen, Tag, Vfy) is (W/S)UF-CMA secure (weakly/strongly unforgeable under chosen message attack) if there is no efficient adversary that wins the relevant game with a significant probability.
- More formally, we say that a MAC scheme is $(q_t, q_v, t, \varepsilon)$-WUF-CMA secure if no adversary making q_t queries to its tag oracle, q_v queries to it verification oracle, and running in time at most t, has success probability greater than ε in the weak unforgeability game.
- Similarly for $(q_t, q_v, t, \varepsilon)$-SUF-CMA security.
- WUF-CMA is also called EUF-CMA ("existential unforgeability") or sometimes just UF-CMA.

WUF-CMA vs SUF-CMA security

- A successful WUF-CMA adversary also breaks SUF-CMA security.
- Hence SUF-CMA security of a MAC scheme implies its WUF-CMA security.
- However, there exist (deterministic) MAC schemes that are WUF-CMA secure but **not** SUF-CMA secure.
- So WUF-CMA security does not imply SUF-CMA security.

Example: suppose MAC scheme (KGen, Tag, Vfy) is WUF-CMA secure; define a new scheme (KGen, Tag′, Vfy′) with (t+1)-bit tags as follows:

Tag′(K,m): compute $\tau = \mathrm{Tag}(K,m)$; output $0 \,||\, \tau$.

Vfy′(K,m,τ'): parse $\tau' = b \,||\, \tau$ where b is a bit; output $\mathrm{Vfy}(K,m,\tau)$.

Claim: (KGen, Tag′, Vfy′) is WUF-CMA secure but not SUF-CMA secure.

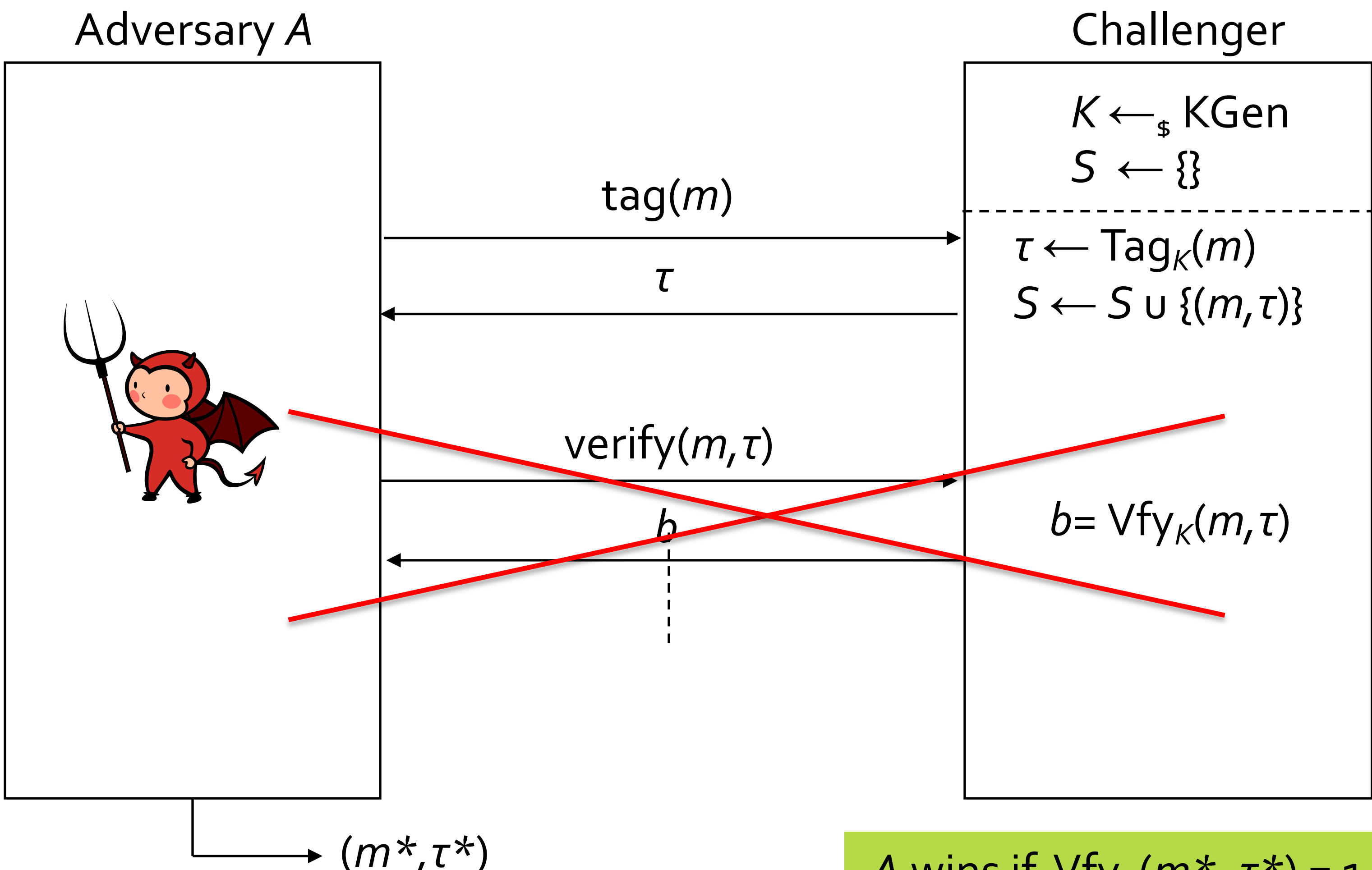
WUF-CMA vs SUF-CMA security

- WUF-CMA and SUF-CMA security are **equivalent** when Tag is deterministic and Vfy is built using Tag (as per slide 7).
- This is because, for any m and K, there is precisely one value τ for which Vfy(K,m,τ)=1, so the extra wins that a SUF-CMA adversary might have beyond those of a WUF-CMA adversary cannot arise.
- Proof: DIY!

Generic Attacks on MACs

- An adversary can always guess the k-bit key K: success prob. 2^{-k}.
- An adversary can use a few pairs (m,τ) obtained from its tag oracle to enable an exhaustive search for the key.
- An adversary that makes q_v queries verify(m, τ) for random values of τ will win with probability at least $q_v \cdot 2^{-t}$.
- Tags and keys need to be long enough to withstand these generic attacks, and q_v needs to be limited to keep $q_v \cdot 2^{-t}$ to a desired level.
    - In typical applications, for the adversary to make a verify query in practice, it needs to interact with some party, sending pairs (m, τ).
    - That is, attacks involving verify queries are inherently **online** and receiver can then limit the number of invalid MAC tags it accepts.
    - Often q_v can be kept quite small; e.g. $q_v = 1$ for TLS.
    - In other cases, e.g. QUIC or DTLS, many forgery attempts are possible and q_v has to be carefully limited.

Alternative SUF-CMA security model: no verification queries



A wins if $\mathrm{Vfy}_K(m^*, \tau^*) = 1$ and $(m^*,\tau^*) \notin S$.

Removing verify queries from the model

Theorem:

Let *MAC* = (KGen, Tag, Vfy) be a MAC scheme.

For any $(q_t, q_v, t, \varepsilon)$-SUF-CMA adversary *A* against *MAC*, there is a $(q_t, t', \varepsilon/q_v)$-no-verify-SUF-CMA adversary *B* against *MAC*, with $t' \approx t$.

Proof:

See extra slides

Impact:

We can work in a simplified security model with only tag queries.

NB1: this result is not true for WUF-CMA: there are (artificial) MAC schemes which are WUF-CMA-secure if q_t=1 but where there exists an efficient WUF-CMA adversary with advantage 1 if $q_t > 1$!

NB2: the result *is* still true for WUF-CMA security when Tag is deterministic and Vfy is built using Tag (as per slide 7).

See https://eprint.iacr.org/2004/309.pdf for all the details!

Attacks that MACs do not prevent

Caution!!!

- Secure MACs do provide integrity and data origin authentication.
- MACs cannot detect message deletion, replay, or reordering attacks.
  - Additional mechanisms (e.g. sequence numbers) are needed in conjunction with MACs to provide these services.
- If the same MAC key K is used in both communication directions, then MACs cannot prevent reflection attacks.
  - Here, a pair (m,τ) created by A and sent to B is sent *back* to A by the adversary.
  - A accepts (m,τ) as having come from B, when in fact it was generated by A!
  - Can prevent such attacks using directional indicators encoded in messages, or using *key separation* (different keys for different purposes).
- MACs do not provide any kind of confidentiality guarantees.
  - Need to use them in careful combination with encryption schemes to get both integrity and confidentiality.

MACs from PRFs

MACs from PRFs

Construction:

Let F: $\{0,1\}^k \times \mathcal{X} \longrightarrow \{0,1\}^n$ be a function.

We build a MAC scheme MAC(F) from F as follows:

KGen: $K \leftarrow_{\$} \{0,1\}^k$

Tag(K,m): $\tau \leftarrow F(K,m)$

Vfy(K,m,τ): $\tau' \leftarrow F(K,m)$; if $\tau' = \tau$ return 1; else return 0.

- This MAC scheme can handle input messages from set $\mathcal{X}$, the input domain of F (typically some set of bit-strings).
- Tag length = output length of F (denoted n here).
- Deterministic Vfy built using "standard method", so the scheme has unique MAC tags.

MACs from PRFs

Theorem:

Let F: $\{0,1\}^k \times \mathcal{X} \longrightarrow \{0,1\}^n$ be a function. Consider the MAC scheme MAC(F) built from F as on the previous slide.

For any (q_t, t, ε)-SUF-CMA adversary A against MAC(F), there exists an adversary B against PRF security of F that runs in time $t' \approx t$, makes q_t+1 oracle queries, and has advantage at least $\varepsilon - (1/2^n)$.

Informally: MAC(F) is SUF-CMA secure under the assumption that F is a PRF.

NB1: we use the "no verify oracle" version of SUF-CMA security.
NB2: because the scheme uses the "standard" verify algorithm (as per slide 7), it actually suffices to prove "*no-verify-oracle-WUF-CMA security*".

MACs from PRFs

Proof:

We proceed via a sequence of two games.

G_0: The no-verify-WUF-CMA security game: A's tag($\cdot$) queries are answered using $F(K,\cdot)$ for a random choice of key K.

Event W_0: A outputs m^* (distinct from tag queries $m_1, \ldots, m_{qt}$) and τ^* such that $\tau^* = F(K, m^*)$, i.e. A breaks SUF-CMA security.

G_1: The no-verify-WUF-CMA security game, but A's tag queries are answered using a random function f.

Event W_1: A outputs m^* (distinct from tag queries $m_1, \ldots, m_{qt}$) and τ^* such that $\tau^* = f(m^*)$.

We will show two things:

1. The difference in probabilities of events W_i in games G_i can be bounded by the advantage of a related PRF adversary B.
2. $\Pr[W_1]$ is small, exactly $1/2^n$.

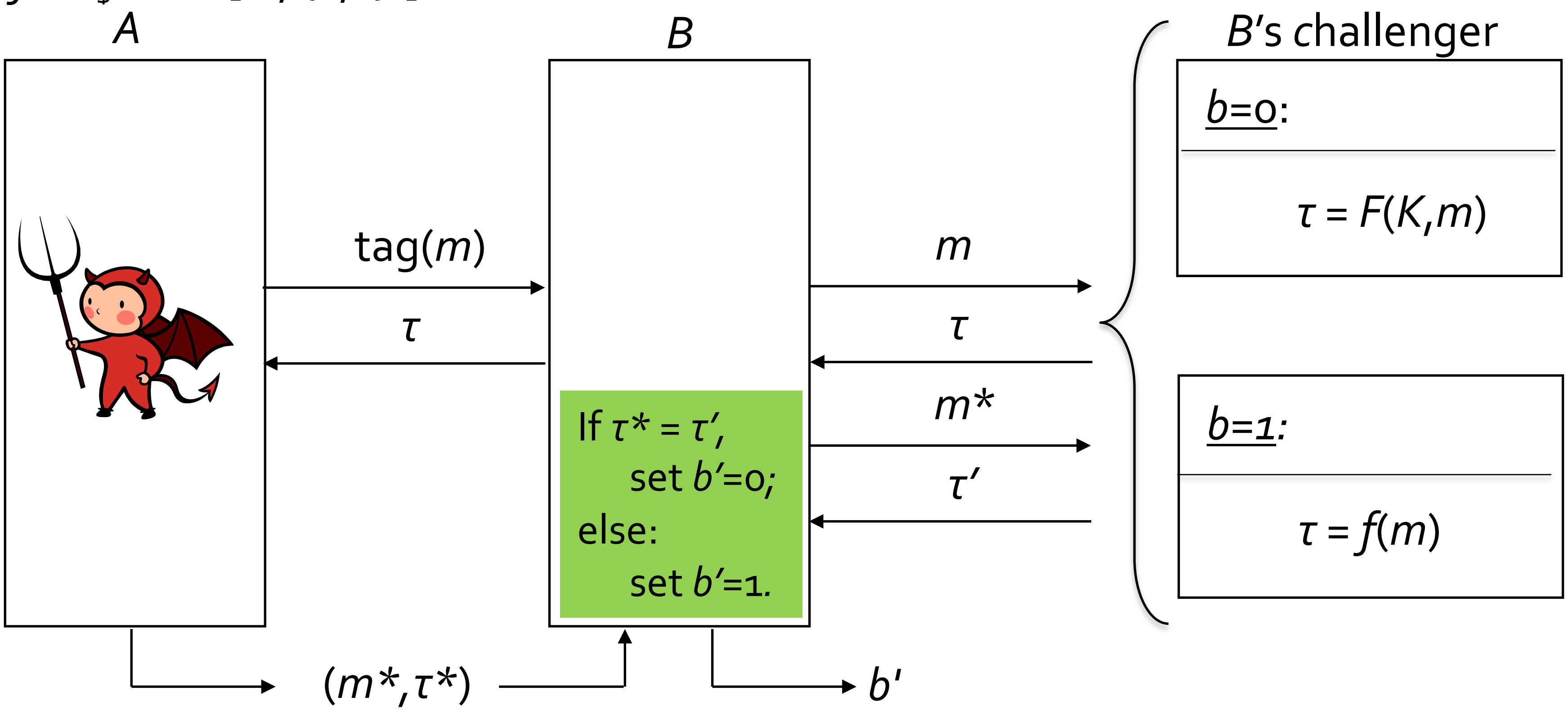
G_0 and G_1, and adversary B

Setup:

$b \leftarrow_{\$} \{0,1\}$

$K \leftarrow_{\$} \{0,1\}^k$

$f \leftarrow_{\$} \text{Funcs}[\mathcal{X}, \{0,1\}^n]$



MACs from PRFs

- B is a PRF adversary making q_t+1 queries, running in same time as A.
- When b=0, A plays in G_0: the no-verify-WUF-CMA security game.
- When b=1, A plays in G_1: the no-verify-WUF-CMA security game but where tag queries are handled using outputs of random function f.

Claim 1: $|\Pr[W_0] - \Pr[W_1]| = \mathrm{Adv}_F^{PRF}(B)$.

Claim 2: $\Pr[W_1] = 1/2^n$.

Combining the claims:

$$\begin{aligned}\mathrm{Adv}_{\mathrm{MAC}(F)}^{WUF\text{-}CMA}(A) = \Pr[W_0] &= |(\Pr[W_0] - \Pr[W_1]) + \Pr[W_1]| \\ &\leq |\Pr[W_0] - \Pr[W_1]| + \Pr[W_1] \\ &= \mathrm{Adv}_F^{PRF}(B) + 1/2^n\end{aligned}$$

MACs from PRFs

Claim 1: $|\Pr[W_0] - \Pr[W_1]| = \mathrm{Adv}_F^{PRF}(B)$.

Proof:

$$\mathrm{Adv}_F^{PRF}(B) = |\Pr[b'=0 \mid b=0] - \Pr[b'=0 \mid b=1]|$$

$$= |\Pr[\tau^* = \tau' \mid A \text{ plays in } G_0] - \Pr[\tau^* = \tau' \mid A \text{ plays in } G_1]|$$

$$= |\Pr[\tau^* = F(K,m^*) \mid A \text{ plays in } G_0] - \Pr[\tau^* = f(m^*) \mid A \text{ plays in } G_1]|$$

$$= |\Pr[W_0] - \Pr[W_1]|$$

MACs from PRFs

Claim 2: $\Pr[W_1] = 1/2^n$.

Proof:

Recall the definition of event W_1:

> "A outputs m^* (distinct from tag queries $m_1,\ldots, m_{qt}$) and τ^* such that $\tau^* = f(m^*)$."

For W_1 to occur, A must output the value of f on a fresh input m^*, having seen the value of f on chosen inputs $m_1,\ldots, m_{qt}$.

But f is a random function, so the value of f at m^* is uniformly random and independent from its value on all other inputs.

Hence $\Pr[W_1] = 1/2^n$ (since f has n-bit outputs).

MAC domain extension from CR hashing

MAC domain extension from CR hashing

- Previous construction gives us a MAC scheme on message input domain $\mathcal{X}$, the input space of the PRF *F*.
- For example, if *F* is instantiated with a block cipher, then $\mathcal{X}$ will be $\{0,1\}^n$ where *n* is the block size in bits, and also now the tag size.
- This is restrictive – our original definition of MACs says $\mathcal{X} = \{0,1\}^*$!

- **Domain extension**: build a new MAC with a larger input domain $\mathcal{X}'$ from a MAC with a fixed input domain $\mathcal{X}$.
- We do this by introducing a collision resistant hash function *H* mapping $\mathcal{X}'$ to $\mathcal{X}$.

MAC domain extension from CR hashing

Theorem:

Let *MAC* = (KGen,Tag,Vfy) be a MAC scheme for message input space $\mathcal{X}$ with tag-length *t* and key-length *k*.

Let $H: \mathcal{X}' \to \mathcal{X}$ be a hash function.

Define a new MAC scheme *HtMAC* = (KGen, Tag′, Vfy′) for message input space $\mathcal{X}'$ by:

Tag′(*K*,*m*): output Tag(K,H(m))

Vfy′(*K*,*m*,*τ*): output Vfy(K,H(m),τ).

If *MAC* is SUF-CMA secure and *H* is collision-resistant, then *HtMAC* is SUF-CMA secure (and has message input space $\mathcal{X}'$).

MAC domain extension from CR hashing

Proof (of domain extension theorem):
Let A be a (q_t, t, ε)-SUF-CMA adversary against *HtMAC*.

We construct from A a SUF-CMA adversary B against *MAC* and a CR adversary C against H such that:

$$\mathrm{Adv}_{HtMAC}^{SUF\text{-}CMA}(A) \leq \mathrm{Adv}_{MAC}^{SUF\text{-}CMA}(B) + \mathrm{Adv}_{H}^{CR}(C)$$

Moreover, both B and C run in time (roughly) t, and B makes q_t queries to its tag(·) oracle.

NB: as usual we work with the "no-verify-oracle" version of SUF-CMA security.

MAC domain extension from CR hashing

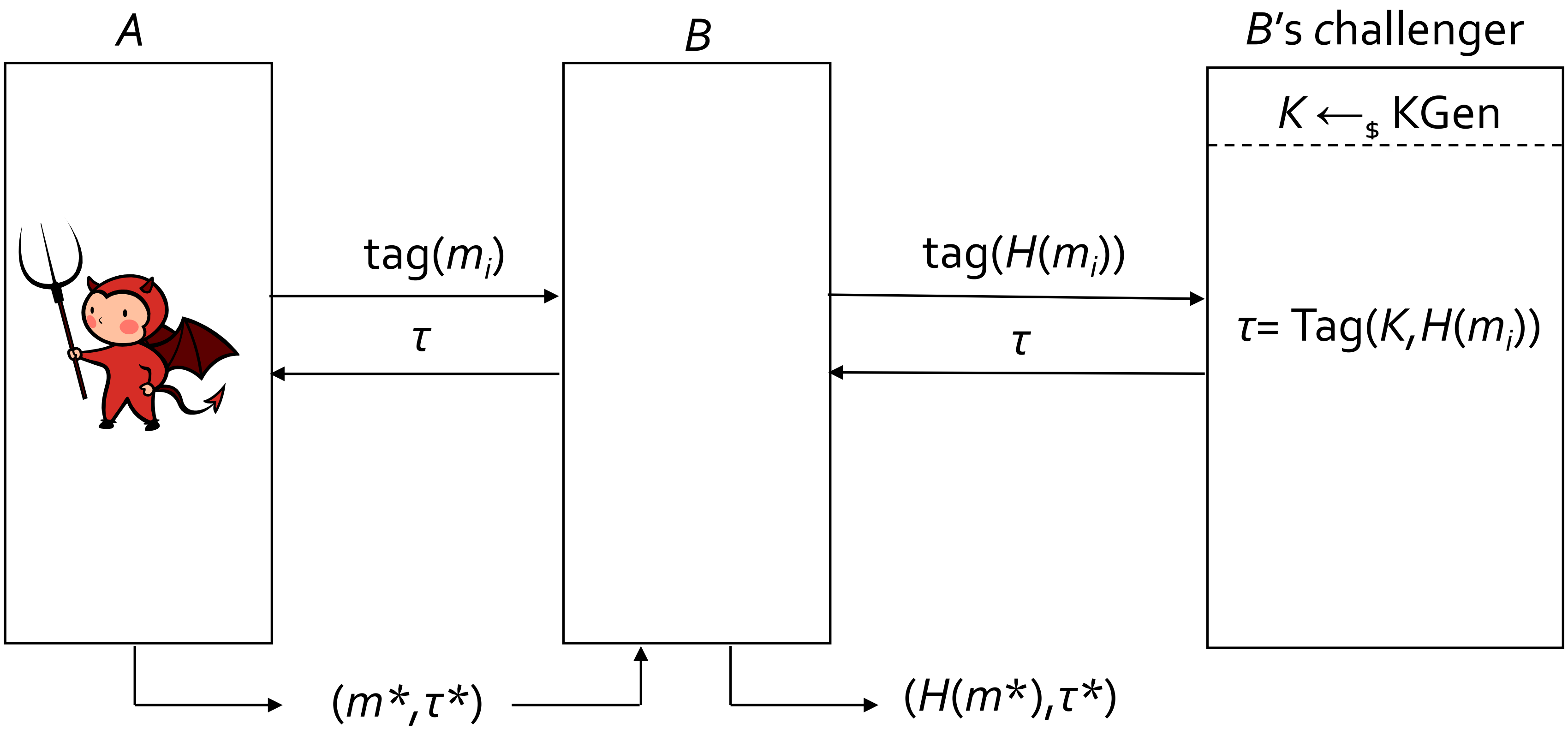
Proof (of domain extension theorem, ctd):

- Let X be the event that adversary *A* wins the SUF-CMA security game against *HtMAC*; let *A*'s output (forgery) be (m^*, τ^*).
- Let Y denote the event that $H(m^*) = H(m_i)$ for some m_i queried by *A* to its tag oracle, with $m^* \neq m_i$.
- Let Z denote the event that *A* wins the SUF-CMA security game against *HtMAC* but event Y does not occur, i.e. $Z = X \wedge \neg Y$.
- We have:

$$\begin{aligned} \mathrm{Adv}^{SUF\text{-}CMA}_{HtMAC}(A) = \Pr[X] &= \Pr[X \wedge \neg Y] + \Pr[X \wedge Y] \\ &\leq \Pr[Z] + \Pr[Y] \end{aligned}$$

- We now construct *B*, a SUF-CMA adversary against *MAC* with advantage *at least* Pr[Z]; and *C*, a CR adversary for *H* with advantage exactly Pr[Y].

Construction of *B*



Key observations: suppose Z = X ∧ ¬Y occurs.

- Event X implies *A*'s output (m^*,τ^*) is a valid forgery for *HtMAC*.
- Event ¬Y implies $H(m^*)$ distinct from $H(m_i)$ for all *i*.
- In combination, *B*'s output $(H(m^*),\tau^*)$ is then a valid forgery for *MAC*.
- So *B* breaks *MAC* in the SUF-CMA sense whenever Z occurs.

Construction of C

A

C

$K \leftarrow_{\$} \mathsf{KGen}$

$\mathsf{tag}(m_i)$ →

← τ_i

$\tau = \mathsf{Tag}(K, H(m_i))$

If $H(m^*) = H(m_i)$ and $m^* \neq m_i$ for some i then output (m^*, m_i).

(m^*, τ^*)

Key observation: C is successful in breaking CR of H iff Y occurs.

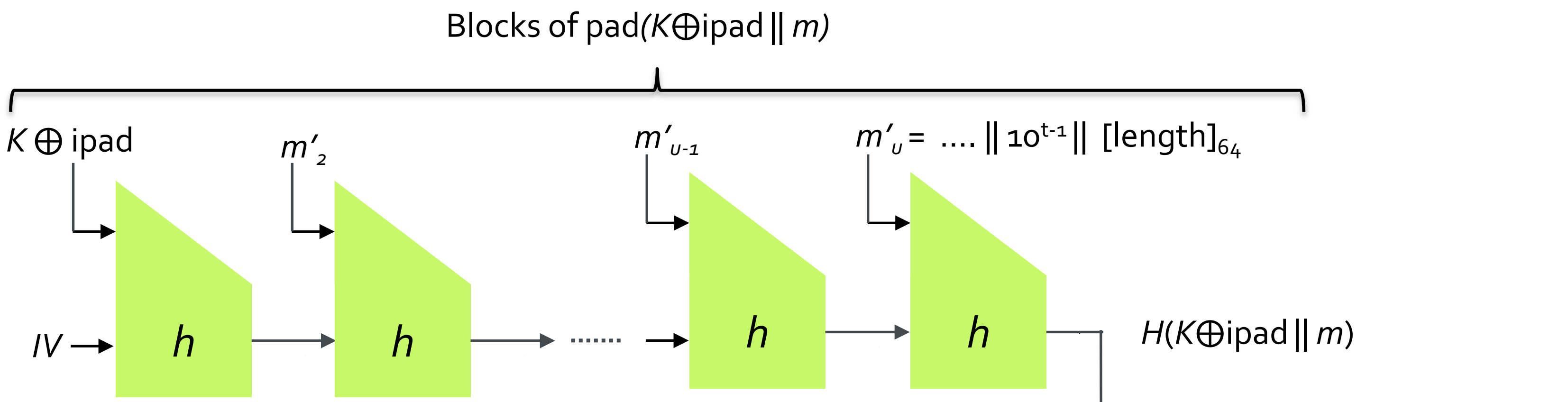
HMAC

HMAC

- MD-based hash functions like SHA-256 are very fast in software.
- Can we build a MAC from such a hash function H?
- Have seen Hash-then-MAC approach, relies on CR of H, vulnerable to offline collision-finding attacks.
- Can we do better?
- We want to turn an unkeyed primitive (SHA-256, say) into a keyed primitive (a MAC) whilst relying on a weaker assumption than CR.
- Prepending the key: length extension attacks.
- Appending the key: vulnerable to *offline* collision attacks on H.
- HMAC uses a two-key nest to build a PRF F_{nest}:

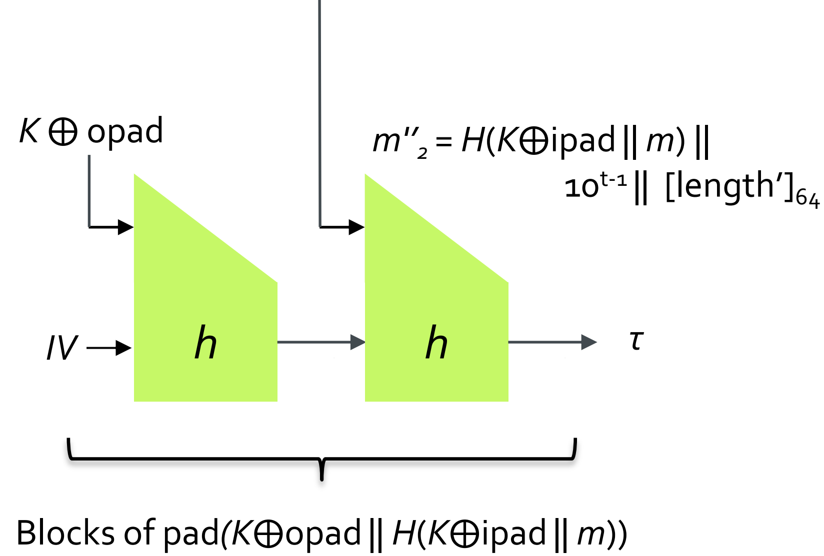
$$F_{\text{nest}}((K_1,K_2),M) := H(K_2 \,\|\, H(K_1 \,\|\, M)).$$

HMAC



ipad: = 0x36.... 0x36
opad: = 0x5C..... 0x5C

NB1: key K is extended to exactly k bits by 0x00 padding before XOR with ipad/opad.
NB2: k is assumed large enough compared to n that hash output $H(K\oplus\text{ipad}\,||\,m)$ + padding + length field fits in a single block.
NB3: length encoding fixed to 64 << k bits.



HMAC – Security

- HMAC uses a two-key nest to build a PRF F_{nest}:

$$F_{\text{nest}}((K_1,K_2),M) := H(K_2 \,\|\, H(K_1 \,\|\, M)).$$

- But keys are derived from a single key K using XOR masks.
- We cannot hope to prove security under a standard assumption on H or compression function h.
- The related NMAC construction, in which independent keys K_1, K_2 are used can be analysed more easily, but the analysis is still non-trivial, and we omit it here.
  - Details of NMAC in Boneh-Shoup, Section 6.5.1.2 and relation between NMAC, two-key nest, and HMAC in Boneh-Shoup, Sections 8.7.1 and 8.7.2.
  - We need h, the compression function, to be a **dual-PRF**: a PRF when either input ("message" and "chaining variable") is regarded as the key and the other input is regarded as the message.
  - Section 8.7.3 of Boneh-Shoup gives a proof of this in the ideal cipher model when h is constructed using Davies-Meyer.
- **Main take-away: under some plausible assumptions, we can assume HMAC is a PRF, and therefore a SUF-CMA MAC.**

HMAC – Usage

- HMAC is specified in RFC 2104, see: https://tools.ietf.org/html/rfc2104.
- HMAC is very widely deployed, especially in IETF protocols like SSL/TLS, IPsec.
- Typically instantiated with MD5 (!), SHA-1, and SHA-256, sometimes with truncation of final output.
- e.g. implicit part of the single “mandatory to implement” cipher suite TLS_AES_128_GCM_SHA256 in TLS 1.3.
- HMAC is gradually being supplanted by faster designs based on universal hashing (seen next section) – e.g. GMAC in GCM, Poly1305 in ChaCha20Poly1305.
- HMAC will still be widely used in key-derivation applications (exploiting PRF-ness rather than SUF-CMA security), e.g. in HKDF.

HMAC – The IUF interface

- Implementations in crypto libraries typically provide HMAC via an "IUF" API.
- The computation is first initialised (I) – this processes $K \oplus$ ipad via compression function h to make first internal chaining value.
- Then the computation is updated (U) as many times as are needed – each call to U can absorb new message bytes via calls to the compression function h.
- When the complete message has been processed, a finalisation (F) step is performed – this processes any last message bytes, completes the inner hash computation, and then does outer hash computation.
- Note that no compression function calls will be made until enough bytes have been received for a compression function call to be needed (or until finalisation is called).
- The IUF interface supports *streaming* applications.
- This design opens up some interesting side-channel opportunities, see for example:

  Martin R. Albrecht, Kenneth G. Paterson:
  Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS. *EUROCRYPT (1) 2016: 622-643.*
  https://eprint.iacr.org/2015/1129

MACs from universal hashing

Introducing nonce-based cryptography

Definition (Nonce)
A nonce is a number-used-once.

Notes:
- "Used once" means *within the context of some randomly sampled symmetric key*.
- Specific constructions of cryptosystems that lie ahead make use of nonces: MAC schemes and Symmetric Encryption (SE) scheme.
- Nonces do not have to be kept secret and do not need to be unpredictable.
- They replace the need for randomness in cryptographic primitives.
- Nonces should in theory be easier for developers to manage than randomness.

Introducing nonce-based cryptography

Definition (Nonce)

A nonce is a number-used-once.

Notes:

- Nonces may be transmitted along with MAC tags, ciphertexts, etc, but nonces can also be *implicit* if they can be inferred from context.
- Nonces are often implemented using counters – but this brings the need for state management
- We could also use random values for nonces (but then birthday bounds apply, and randomness failures may lead to nonce reuse).
- Typically, no security guarantees if any nonce value is used more than once (with the same key).
- This should not happen, but it does sometimes, due to implementation errors or "cryptographic misuse".

Nonce-based MACs

Definition (Nonce-based MAC)

A nonce-based MAC scheme with key length k, nonce space $\mathcal{N}$, and tag length t consists of a triple of efficient algorithms (KGen, Tag, Vfy) such that:

$$\text{KGen}: \{\} \longrightarrow \{0, 1\}^k,$$
$$\text{Tag}: \{0, 1\}^k \times \mathcal{N} \times \{0, 1\}^* \longrightarrow \{0, 1\}^t,$$
$$\text{Vfy}: \{0, 1\}^k \times \mathcal{N} \times \{0,1\}^* \times \{0,1\}^t \longrightarrow \{0,1\}$$

Correctness:

For all $K \in \{0,1\}^k$, $N \in \mathcal{N}$, $m \in \{0,1\}^*$, if

$$\tau \leftarrow \text{Tag}(K,N,m),$$

then

$$1 \leftarrow \text{Vrf}(K,N,m,\tau).$$

Security for nonce-based MACs

A

All nonces N_i must be distinct in A's queries

Challenger

$K \leftarrow_\$ \text{Kgen}$

$S \leftarrow \{\}$

$\text{tag}(N_i, m_i)$ →

← τ_i

$\tau_i \leftarrow \text{Tag}(K, N_i, m_i)$

$S \leftarrow S \cup \{(N_i, m_i, \tau_i)\}$

A → (N^*, m^*, τ^*)

A **wins** if $\text{Vfy}(K, N^*, m^*, \tau^*) = 1$ and $(N^*, m^*, \tau^*) \notin S$

Nonce-based MACs

Definition (Nonce-based MAC security)

- We write $\mathrm{Adv}_{NMAC}^{SUF\text{-}CMA}(A)$ for the winning probability of adversary A in the preceding game against some nonce-based scheme $NMAC$ = (KGen, Tag, Vfy).
- We say that a nonce-based MAC scheme is (q_t,t,ε)-SUF-CMA secure if no adversary making q_t queries to its tag oracle and running in time at most t has success probability greater than ε in the unforgeability game.
- We can also consider a version of the game with a verify oracle(·).
- Note that we give A **complete control over nonces and messages** in queries tag(N_i,m_i), but require nonces to be distinct.
- In reality, some entity is running the Tag algorithm.
- This entity chooses the nonces and must enforce distinctness.

Keyed hash functions and universal hashing

- Our objective is to build very fast nonce-based MACs with good security bounds.
- The introduction of nonces will enable the use of lightweight components and better bounds than otherwise possible.

Definition (Keyed hash function)

A keyed hash function H is a deterministic algorithm that takes two inputs, a key K and a message m; its output $t := H(K,m)$ is called a digest. As usual, there are associated spaces: the keyspace $\mathcal{K}$, the message space $\mathcal{M}$ and the digest space $\mathcal{T}$.

Universal hashing

Definition (Universal hash function security game)
For a keyed hash function H with keyspace $\mathcal{K}$, message space $\mathcal{M}$ and digest space $\mathcal{T}$, and a given adversary A, the game runs as follows:

1. The challenger sets $K \leftarrow_{\$} \mathcal{K}$.
2. A outputs two distinct messages $m_0, m_1 \in \mathcal{M}$.

Finally, A wins if $H(K,m_0) = H(K,m_1)$.

We define A's advantage, denoted $\mathrm{Adv}_H^{UHF}(A)$ as the probability that A wins the above game.

We say that H is an ε-bounded universal hash function, or ε-UHF, if $\mathrm{Adv}_H^{UHF}(A) \leq \varepsilon$ for all adversaries A (even computationally unbounded ones).

Universal hashing – alternative security definition

If H is a keyed hash function with keyspace $\mathcal{K}$, message space $\mathcal{M}$ and digest space $\mathcal{T}$, then H is said to be an ε-UHF if the following holds:

For every pair of distinct messages $m_0, m_1 \in \mathcal{M}$ we have:

$$\Pr[H(K,m_0) = H(K,m_1)] \leq \varepsilon,$$

where the probability is over the random choice of $K \in \mathcal{K}$.

This definition is equivalent to the previous one, and we will use it from now on.

Equivalence:
A is unbounded but has no information about *K*, so an optimal *A* outputs a pair of messages maximising the collision probability over random choice of *K*.

Universal hashing from polynomials

Let $\mathcal{F}$ be a finite field (e.g. integers mod p or GF(2^n)).
Set $\mathcal{K} = \mathcal{T} = \mathcal{F}$, $\mathcal{M} = (\mathcal{F})^{\leq L}$ (i.e. messages are vectors of length at most L over $\mathcal{F}$).

Define a hash function H_{poly} as follows:

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

- Note that $H_{\text{poly}}(K, (a_1,\ldots,a_v))$ is just equal to $a(K)$, where $a(X)$ is a degree v polynomial derived from $(a_1,\ldots,a_v)$, evaluated over $\mathcal{F}$.
- Fast evaluation of H is possible using finite field operations and Horner's rule.

Universal hashing from polynomials

Theorem:

H_{poly} is an ε-UHF for $\varepsilon = L \,/\, |\mathcal{F}|$.

Proof:

- Consider any two distinct inputs $(a_1,\ldots,a_u)$, $(b_1,\ldots,b_v)$ of lengths u, v, respectively.
- Recall that: $H_{\text{poly}}(K, (a_1,\ldots,a_u)) = a(K)$; $H_{\text{poly}}(K, (b_1,\ldots,b_v)) = b(K)$
- Since the inputs are distinct, the polynomials a, b are distinct and so $a - b$ is a non-zero polynomial of degree at most L. (The leading term K^v takes care of message vectors beginning with "zero" components.)
- Hence $a - b$ has at most L roots over $\mathcal{F}$.
- So, for random K, we have: $\Pr[(a - b)(K) = 0] \leq L \,/\, |\mathcal{F}|$.
- So: $\Pr[a(K) = b(K)] \leq L \,/\, |\mathcal{F}|$.
- Hence: $\Pr[H_{\text{poly}}(K, (a_1,\ldots,a_u)) = H_{\text{poly}}(K, (b_1,\ldots,b_v))\,] \leq L \,/\, |\mathcal{F}|$.

Building a PRF from a PRF + universal hashing

Construction (UHFtPRF) composition):

Suppose H is an ε-UHF with keyspace $\mathcal{K}$, message space $\mathcal{M}$ and digest space $\mathcal{T}$, and F is a secure PRF with keyspace $\mathcal{K}'$, message space $\mathcal{T}$ and output space $\mathcal{X}$. Define a function F' by:

$$F'((K_1, K_2), m) := F(K_2, H(K_1, m)).$$

Theorem:

F' is a secure PRF, for keyspace $\mathcal{K} \times \mathcal{K}'$, message space $\mathcal{M}$ and output space $\mathcal{X}$.

In particular, suppose A is a PRF adversary against F' making at most q queries. Then there exist a PRF adversary B against F (also making q queries) such that:

$$\mathrm{Adv}_{F'}^{PRF}(A) \leq \mathrm{Adv}_{F}^{PRF}(B) + (q^2/2) \cdot \varepsilon.$$

Difference unpredictable hashing and XOR universal hashing

Definition (difference unpredictable hash function security game)
For a keyed hash function H with keyspace $\mathcal{K}$, message space $\mathcal{M}$ and digest space $\mathcal{T}$ equipped with a group operation "+" (and inverse "–"), and a given adversary A, the attack game runs as follows:

1. The challenger picks a random $K \leftarrow_{\$} \mathcal{K}$.
2. A outputs two distinct messages $m_0, m_1 \in \mathcal{M}$ and a value $\delta \in \mathcal{T}$.

Finally, A wins if $H(K,m_0) - H(K,m_1) = \delta$.

We define A's advantage, denoted $\mathrm{Adv}_H^{DUHF}(A)$ as the probability that A wins the above game.

We say that H is an ε-bounded difference unpredictable hash function, or ε-DUHF, if $\mathrm{Adv}_H^{DUHF}(A) \leq \varepsilon$ for all adversaries A (inc. comp. unbounded ones).

Difference unpredictable hashing and XOR universal hashing

Typical group operations:

- $\mathcal{T} = \mathbb{Z}_N$, the integers mod N, and "+" being addition mod N.
- $\mathcal{T} = \{0,1\}^n$ for some bit-length n, and "+" replaced with XOR.

In the latter case, we usually refer to ε-XOR-universality instead of ε-bounded difference unpredictability.

Construction: (H_{xpoly})
Let $\mathcal{F}$ be a finite field. Set $\mathcal{K} = \mathcal{T} = \mathcal{F}$, $\mathcal{M} = (\mathcal{F})^{\leq L}$ (so messages are vectors of length at most L over $\mathcal{F}$ and group operation is finite field addition).

Define a hash function H_{xpoly} as follows:

$$H_{\text{xpoly}}(K, (a_1, \ldots, a_v)) := K^{v+1} + a_1 K^v + a_2 K^{v-1} + \cdots + a_{v-1} K^2 + a_v K \in \mathcal{F}$$
$$= K \cdot H_{\text{poly}}(K, (a_1, \ldots, a_v)).$$

Difference unpredictable hashing from polynomials

Theorem:

H_{xpoly} is an ε-DUHF for $\varepsilon = (L + 1) / |\mathcal{F}|$.

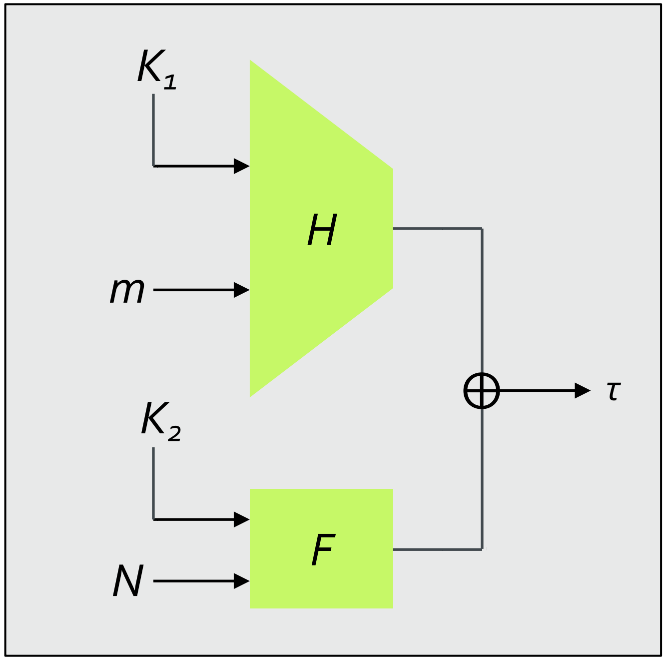
(Hence for large $|\mathcal{F}|$ and reasonable L, we can make ε small.)

Proof:

- Consider any two distinct inputs $(a_1,\ldots,a_u)$, $(b_1,\ldots,b_v)$ of lengths u, v, respectively; let $\delta \in \mathcal{F}$ be arbitrary.
- Now we consider the polynomial $X(a(X) - b(X)) - \delta$.
- It's easy to see that this polynomial is non-zero, has degree at most $L + 1$, and hence has at most $L + 1$ roots.
- Hence, for random K, $\Pr[K(a(K) - b(K)) - \delta = 0] \le (L+1)/|\mathcal{F}|$.
- The rest of the proof is similar to that for H_{poly} and uses the equivalent probabilistic formulation of DUHF security.

Carter-Wegman MACs

- The Carter-Wegman MAC construction combines an ε-DUHF H and a PRF F to build the Tag algorithm.
- Nonce N is input to PRF and produces a pseudorandom value $F(K_2,N)$.
- Hash output $H(K_1,m)$ is combined with $F(K_2,N)$ using the group operation "+" to create MAC tag τ.
- So $F(K_2,N)$ is being used as an encryption mask to "hide" output from weak H.
- This assumes outputs from F can also be interpreted as group elements.
- NB Carter-Wegman is usually presented as a randomised scheme, in which case nonce forms part of tag and tag length is really $|N| + |\tau|$ (cf. shorter tags in UHFtPRF scheme).



Carter-Wegman Tag algorithm

Carter-Wegman MACs

Construction of *CW-MAC(F,H)*:

Let H be an ε-DUHF with outputs in $\mathcal{T}_H$; let F be a PRF on $\{0,1\}^n$ with outputs also in $\mathcal{T}_H$; assume $(\mathcal{T}_H,+)$ is a group. Define the scheme *CW-MAC*(F,H) as follows:

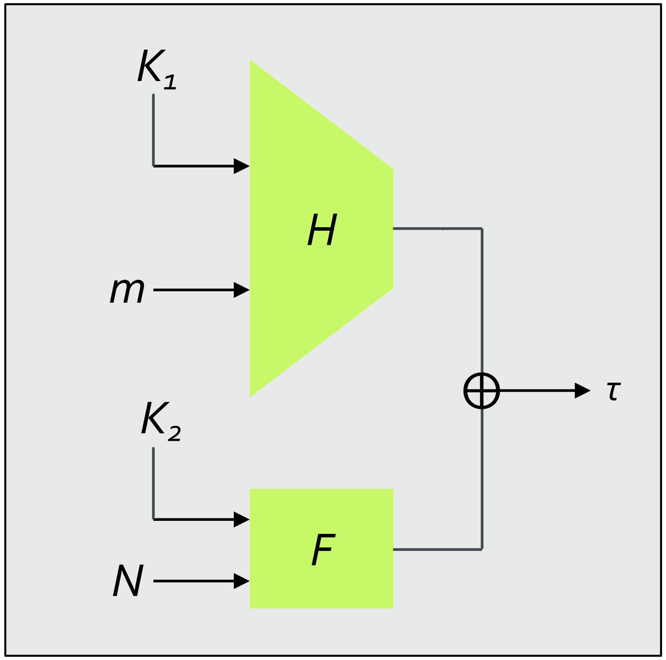
KGen: $(K_1,K_2) \leftarrow_{\$} \mathcal{K}_H \times \mathcal{K}_F$.

Tag: on input (K_1,K_2), nonce $N \in \{0,1\}^n$, message $m \in \mathcal{M}_H$, output:

$$\tau = H(K_1,m) + F(K_2,N).$$

Vfy: on input (K_1, K_2), nonce $N \in \{0,1\}^n$, message $m \in \mathcal{M}_H$, and tag value τ:

1. Set $\tau' = \mathrm{Tag}((K_1,K_2),N,m)$
2. Return 1 if $\tau' = \tau$; 0 otherwise.



Carter-Wegman Tag algorithm

Security of Carter-Wegman MACs

Theorem:

Let H be an ε-DUHF with outputs in $\mathcal{T}_H$; let F be a PRF on $\{0,1\}^n$ with outputs also in $\mathcal{T}_H$; assume $(\mathcal{T}_H,+)$ is a group. Then the scheme *CW-MAC*(F,H) is SUF-CMA secure.

More precisely, for any SUF-CMA adversary A against *CW-MAC*(F,H) making q_t tag queries, there exist a PRF adversary B against F such that:

$$\mathrm{Adv}^{SUF\text{-}CMA}_{CW\text{-}MAC(F,H)}(A) \leq \mathrm{Adv}^{PRF}_{F}(B) + \varepsilon + 1/|\mathcal{T}_H|.$$

Moreover, B runs in time roughly the same as A and makes q_t queries to its oracle.

Security of Carter-Wegman MACs

Proof Sketch:

We use two games.

In G_0 , an adversary A interacts with the real scheme implemented using F_{K2}.

In G_1, we replace all of the outputs of F_{K2} with outputs of a random function f.

A standard analysis shows that the A's winning probability in the two games differs by at most $\mathrm{Adv}_F^{PRF}(B)$, for some PRF adversary B that runs in the same time as A, which makes q_t queries, and whose advantage is the same as A's.

The rest of the analysis is done with A running in G_1; we just need to bound

$$\Pr[A \text{ wins in } G_1].$$

Security of Carter-Wegman MACs

Proof Sketch (ctd):

By assumption, all nonces N in A's queries are distinct. There are two disjoint ways A can win in G_1:

1. Event E_1: A wins and outputs a triple (N^*,m^*,τ^*) in which N^* is **new** (N not used in any of A's tag queries).
2. Event E_2: A wins and outputs a triple (N^*,m^*,τ^*) in which $N^* = N$ with N repeated from some previous tag query.

These events are disjoint, and so:

$$\Pr[A \text{ wins in } G_1] = \Pr[E_1] + \Pr[E_2].$$

Security of Carter-Wegman MACs

Proof Sketch (ctd – case 1):

For A to win with output (N^*, m^*, τ^*) in G_1, we must have:

$$\tau^* = H(K_1, m^*) + f(N^*).$$

Rearranging, we must have:

$$f(N^*) = \tau^* - H(K_1, m^*) \qquad (*)$$

in the relevant group $(\mathcal{T}_H, +)$.

- In case of event E_1, N^* is new.
- But then $f(N^*)$ is uniformly random in $\mathcal{T}_H$ and independent from all the other outputs of f seen by A.
- So when A produces its output, it has no information on the value $f(N^*)$.
- Hence equation (*) holds with probability $1 / |\mathcal{T}_H|$.
- So $\Pr[E_1] = 1 / |\mathcal{T}_H|$.

Security of Carter-Wegman MACs

Proof Sketch (ctd – case 2):

2. Event E_2: A outputs a triple (N^*, m^*, τ^*) in which $N^* = N$ with N repeated from some previous tag query.

Then we have the equations:

$$\tau^* = H(K_1, m^*) + f(N)$$

$$\tau = H(K_1, m) + f(N)$$

- Subtracting (in the group), we get:

$$\tau^* - \tau = H(K_1, m^*) - H(K_1, m)$$

- From *A*'s output, we can then build an adversary *C* that breaks DUHF-security of *H* with output $(m^*, m, \tau^*-\tau)$.
- But the success probability of *any* such adversary is bounded by ε.
- Hence $\Pr[E_2] \leq \varepsilon$.

Instantiations of Carter-Wegman MACs

- Most natural:
  - Instantiate F (the PRF) with AES, apply PRP-PRF switching lemma.
  - Instantiate H with H_{xpoly} over $\mathcal{F} = GF(2^{128})$ and some maximum message length L.
  - This is *close* to the GMAC algorithm that appears in AES-GCM (NIST SP 800-38D).
  - Main difference is that GMAC maps bits to fields elements and uses a message length encoding field instead of the "K^{v+1}" term.
  - Special instructions to support GMAC on Intel and AMD chips: https://en.wikipedia.org/wiki/CLMUL_instruction_set
- Bernstein's Poly1305-AES:
  - Uses AES to instantiate F.
  - Builds on polynomial hashing, with several tweaks for efficiency.
  - Uses $\mathcal{F} = GF(p)$ with $p = 2^{130} - 5$. This choice enables fast modular reduction.
  - Also used (with different PRF) in ChaCha20-Poly1305 Authenticated Encryption.

Extra slides

Removing verification queries from the SUF-CMA model for MAC security

Removing verify queries from the model

Theorem:

Let *MAC* = (KGen, Tag, Vfy) be a MAC scheme.

For any $(q_t, q_v, t, \varepsilon)$-SUF-CMA adversary *A* against *MAC*, there is a $(q_t, t', \varepsilon/q_v)$-no-verify-SUF-CMA adversary B_2 against *MAC*, where $t' \approx t$.

Proof:

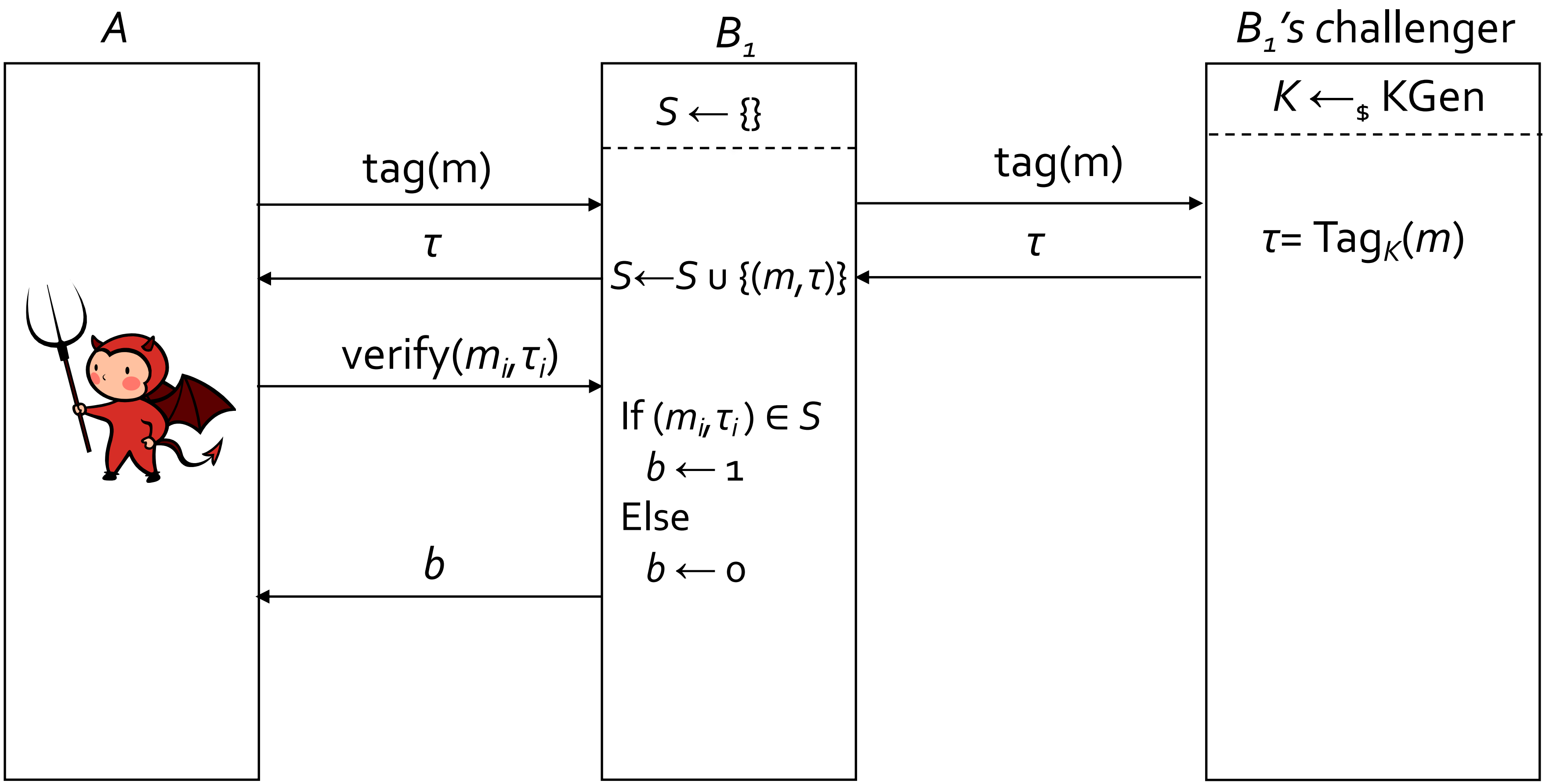
We will use a sequence of 3 games:

$\mathbf{G_0}$: the original SUF-CMA security game with verify queries.

$\mathbf{G_1}$: as G_0, but B_1, a no-verify adversary against *MAC*, relays all *A*'s tag queries to its challenger and responds with "0" (invalid) to all of *A*'s new verify queries.

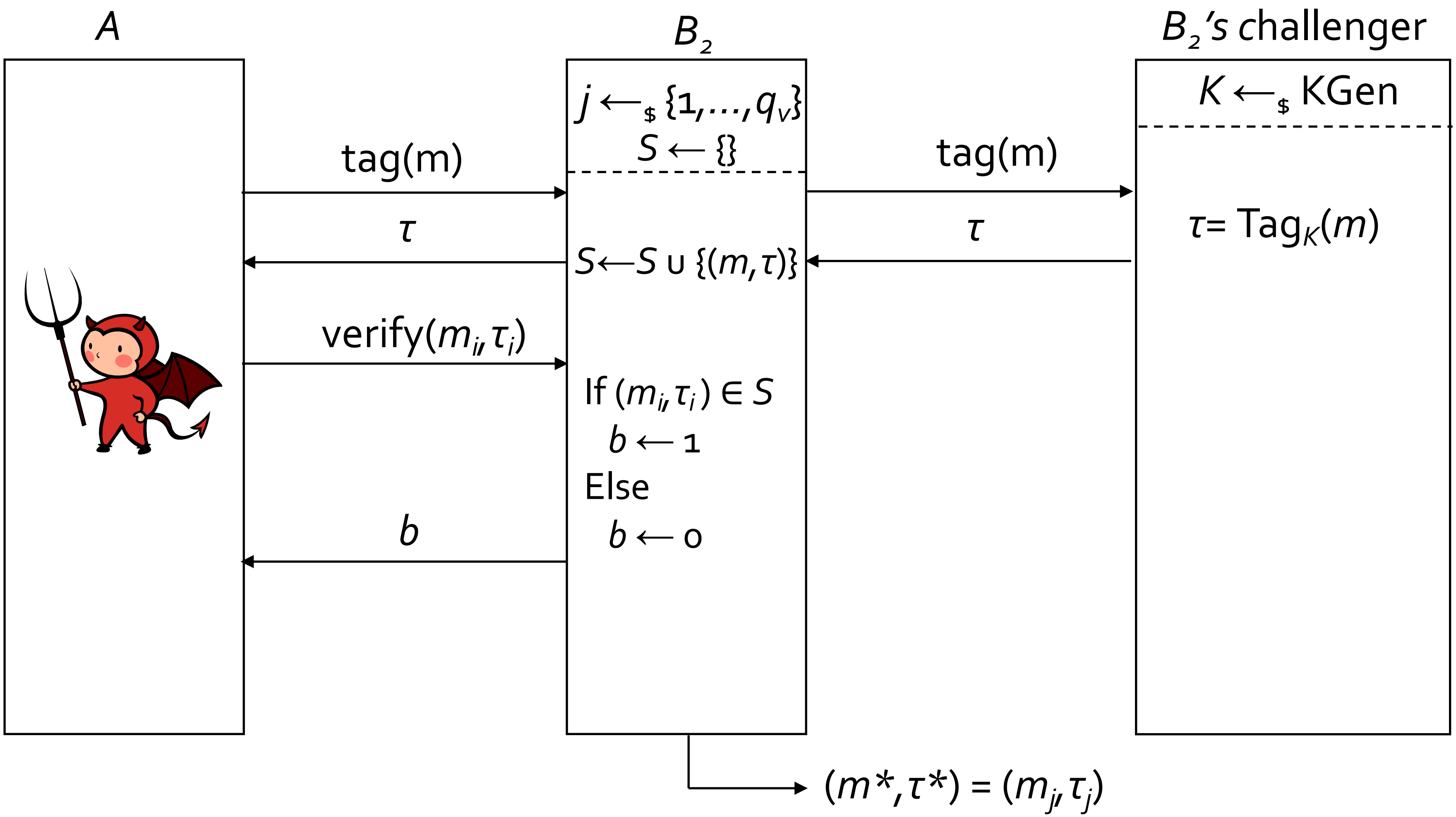
$\mathbf{G_2}$: as G_1, but B_2, a no-verify adversary against *MAC*, relays all *A*'s tag queries to its challenger, responds with "0" to all *A*'s new verify queries, and outputs a random selection of *A*'s verify queries (m^*, τ^*) as its forgery.

G_1



A
B_1
B_1's challenger
$S \leftarrow \{\}$
$K \leftarrow_{\$}$ KGen
tag(m)
tag(m)
τ
τ
$\tau = \mathrm{Tag}_K(m)$
$S \leftarrow S \cup \{(m, \tau)\}$
verify(m_i, τ_i)
If $(m_i, \tau_i) \in S$
$b \leftarrow 1$
Else
$b \leftarrow 0$
b

G_2



A
B_2
B_2's challenger
$j \leftarrow_\$ \{1,\dots,q_V\}$
$S \leftarrow \{\}$
$K \leftarrow_\$$ KGen
tag(m)
tag(m)
τ
τ
$\tau = \mathrm{Tag}_K(m)$
$S \leftarrow S \cup \{(m,\tau)\}$
verify(m_i,τ_i)
If $(m_i,\tau_i) \in S$
$b \leftarrow 1$
Else
$b \leftarrow 0$
b
$(m^*,\tau^*) = (m_j,\tau_j)$

Removing verify queries from the model

G_0: the original SUF-CMA security game with verify queries.

G_1: as G_0, but *B* relays all *A*'s tag queries to its challenger and responds with "0" (invalid) to all of *A*'s new verify queries.

- Let W_0 denote the event that in G_0, $\mathrm{Vfy}(K,m_i,\tau_i) = 1$ and $(m_i,\tau_i) \notin S$ for *at least one value* of *i* in *A*'s verify(·) queries (so $\Pr[W_0]$ is *A*'s success probability in G_0).
- Let W_1 denote the same event in G_1.
- Let i^* denote the first such value of *i* in G_1 (set $i^* = q_v+1$ if W_1 does not arise).
- i^* is a random variable (with unknown distribution).
- Games G_0 and G_1 proceed identically up until event W_0 (or W_1) occurs at query i^* and so:

$$\Pr[W_1] = \Pr[W_0].$$

- This is despite *A* in G_1 not necessarily receiving the correct verify oracle response for new queries, and despite that the games may "diverge" after W_0 (or W_1) occurs.

Removing verify queries from the model

G_1: as G_0, but B_1 relays all A's tag queries to its challenger and responds with "0" (invalid) to all of A's new verify queries.

G_2: as G_1, but B_2 relays all A's tag queries to its challenger, responds with "0" to all A's new verify queries, **and outputs a random selection of A's verify queries (m_j, τ_j) as its forgery.**

- Note that G_1 and G_2 are identical except for the output of B_1 / B_2.
- Let W_2 denote the event that in G_2, we have $\mathrm{Vfy}(K, m_j, \tau_j) = 1$ and $(m_j, \tau_j) \notin S$ for the random choice of j made by B_2.
- Note that W_2 occurs in G_2 if and only if B_2 is successful, i.e. $\Pr[W_2]$ is B_2's winning probability in the no-verify-oracle version of the SUF-CMA game.
- Note also that if W_1 occurs in G_2 and $j = i^*$ then B_2 is also successful.
- Hence:

$$\Pr[W_2] \geq \Pr[W_1 \wedge (j = i^*)] \geq \Pr[j = i^* \mid W_1] \cdot \Pr[W_1].$$

Removing verify queries from the model

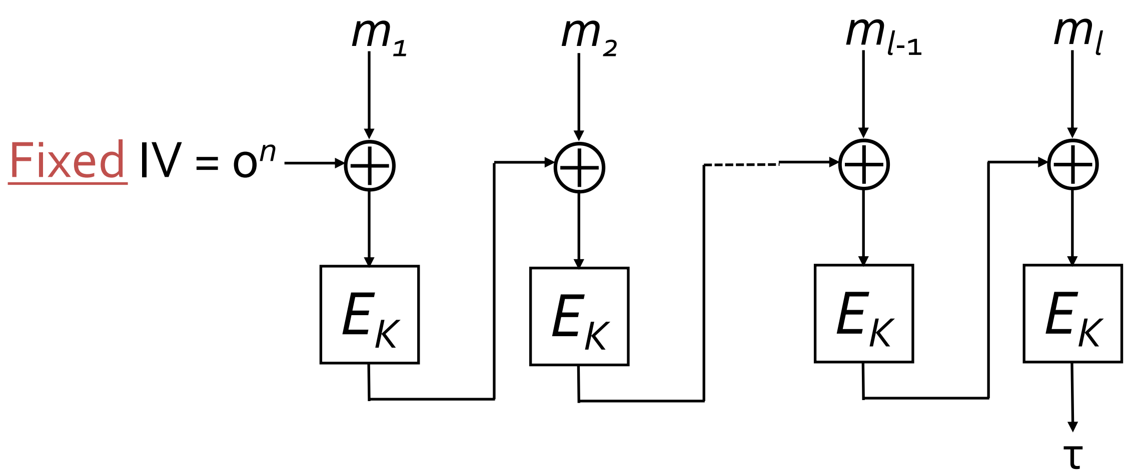
- But $\Pr[j=i^* \mid W_1] = 1/q_v$ (no matter the distribution on i^*).
- This is because j was chosen uniformly at random from $\{1,\ldots,q_v\}$ (independent of everything else in the game) and if W_1 occurs then we know that $1 \leq i^* \leq q_v$.
- We deduce that:

$$\Pr[W_2] \geq \Pr[j=i^* \mid W_1] \cdot \Pr[W_1] = \Pr[W_1] / q_v = \Pr[W_0] / q_v .$$

- Recall that $\Pr[W_2]$ is B_2's winning probability in the no-verify-oracle version of the SUF-CMA game.
- This construction of B_2 and the above bound on its winning probability completes the proof.
- NB proof can be simplified a bit by assuming that A avoids "useless" verification queries.

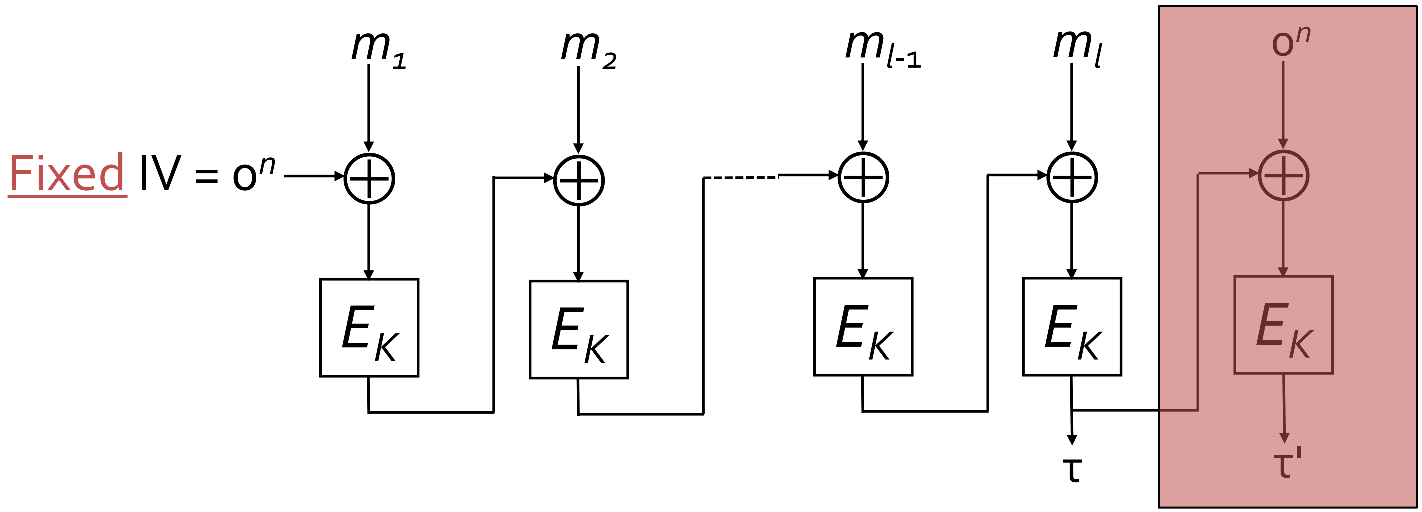
CBC-MAC and CMAC

Basic CBC-MAC



- Let E be an n-bit block cipher with key K.
- Split message m into n-bit chunks $m_1, \ldots, m_l$ (may require padding).
- Compute τ= Tag(K,m) from blocks $m_1, \ldots, m_l$ as in the picture.
- Vfy: standard algorithm (rerun Tag on (K,m) and compare to received tag τ).
- Plausible security based on pseudorandomness of CBC mode.
- Padding pitfalls: avoid zero-bit-padding (why?)

CBC-MAC length extension attack



- If τ = Tag(K, m), then Tag(K,$m\|0^n$) = Tag(K,τ) !
- SUF-CMA attack: obtain τ= Tag(K,m) using one tag oracle call, and then τ' = Tag(K,τ) using a second call.
- Can then output as a MAC forgery the pair ($m\|0^n$, τ'), breaking SUF-CMA security.
- Various other length-extension attacks are possible, see exercises.

CBC-MAC

- However, CBC-MAC is provably SUF-CMA-secure if all messages *m* have the same length.
- We can also resolve length-related security issues using various tweaks:
  - Prefix-free encodings: for example, prepend the message length to the the message before CBC-MAC.
    - Inconvenient for streaming applications: need to know message length before commencing tag computation.
  - Output truncation: shorten τ to prevent "chaining" attacks.
  - Last block encryption: encrypt the last output block of the CBC chain under a separate key.
  - ISO 8731-1 and ISO/IEC 9797: combine OZ-padding + last block encryption + truncation.
  - CBC-MAC is quite widely used in financial industry.
  - Security proofs in Boneh-Shoup; we omit details.

CMAC

- CMAC is a NIST standard for using AES or triple DES in CBC mode to build a MAC (https://csrc.nist.gov/publications/detail/sp/800-38b/final).
- Uses a variant of ECBC (CBC-MAC with last block encryption).
- Avoids addition of full block of padding for block-aligned data by using a two-key approach.
- Significantly more “fiddly” than other CBC-MAC variants we’ve looked at.
- Unclear if the additional complexity is worthwhile – greater likelihood of implementation errors, bugs in security proofs.
- Details follow.

CMAC – Subkey generation

Prerequisites: block cipher E with block size n; key K.

Output: subkeys $K1$, $K2$.

Steps:

1. Let $L = E_K(0^n)$.
2. If MSB1(L) = 0, then $K1 = L \ll 1$; Else $K1 = (L \ll 1) \oplus R_n$.
3. If MSB1($K1$) = 0, then $K2 = K1 \ll 1$; Else $K2 = (K1 \ll 1) \oplus R_n$.
4. Return K1, K2.

Here, for n =128, $R_{128} = 0^{120}10000111$.

NB Steps 2 and 3 correspond to multiplication of L (interpreted as a finite field element) by field elements X and X^2 in $GF(2^n)$.

CMAC – Padding

Input: $m \in \{0,1\}^*$ and subkeys $(K1, K2)$.

Output: block-aligned message $a = a_1 \| \cdots \| a_v$.

Steps:

1. If $|m|$ is not a positive multiple of n then $u \leftarrow |m| \bmod n$.
2. Partition m into a sequence of bit strings $m_1, \ldots, m_v$ so that $m = m_1 \| \cdots \| m_v$ and $m_1, \ldots, m_{v-1}$ are n-bit strings.
3. If $|m|$ is a positive multiple of n then

   output $m_1 \| \cdots \| m_{v-1} \| (m_v \oplus K1)$;

   else

   output $m_1 \| \cdots \| m_{v-1} \| ((m_v \| 10^{n-u-1}) \oplus K2)$.

- Here, we use $K1$ and $K2$ to "pad" the last blocks in different ways, according to whether the message was block-aligned or not.
- This is a trick to avoids adding an extra block of padding in the block-aligned case.
- $K1$ and $K2$ being unknown to adversary makes task of finding colliding messages hard.

CMAC – Tag algorithm

Input: key K for block cipher E, and message $m \in \{0,1\}^*$.

Output: a CMAC tag value τ.

Steps:

1. Generate subkeys $K1$, $K2$.
2. Perform CMAC padding on input m to produce block-aligned message $a = a_1 \| \cdots \| a_v$.
3. Run standard CBC-MAC with key K and block cipher E on message a:

    $w = 0^n$;

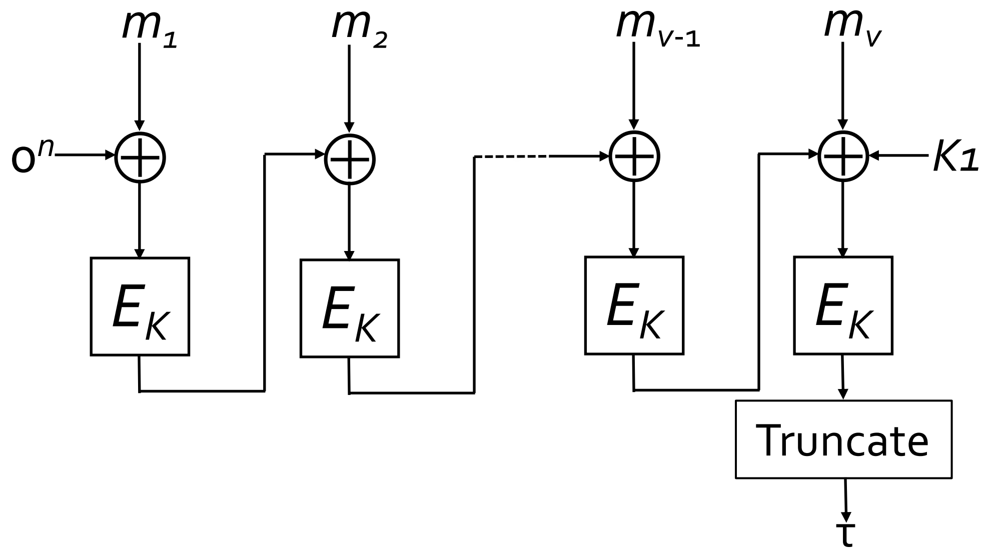
    for $i = 1$ to v do:

        $w = E(K, w \oplus a_i)$

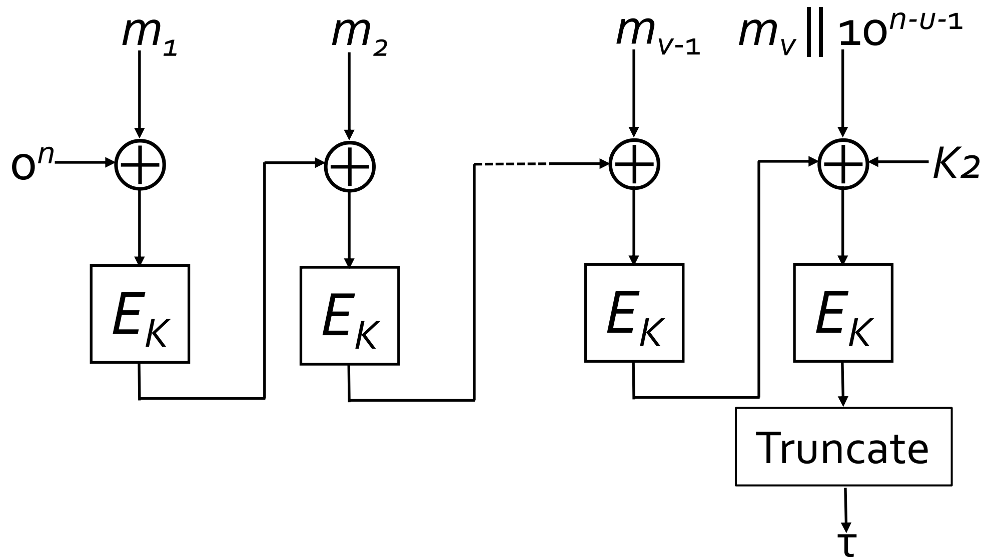
4. Output $\tau = w[0,\ldots,t\text{-}1]$ // t most significant bits of w.

CMAC in a picture

Data block-aligned



Data not block-aligned



CMAC – Implementation and security

- Cost of subkey generation can be amortised over many uses of Tag.
- Then cost is roughly one block cipher call per n bits under a fixed key – fast if AES-NI instructions available.
- Supports streaming-based computation.
- Security analysis under assumption that K, $K1$, $K2$ are independent, random keys is not too difficult: use idea that XORs with $K1$, $K2$ during padding create randomised, prefix-free encodings.
- Full analysis with actual $K1$, $K2$ more complex: see *T. Iwata and K. Kurosawa, OMAC: One-key CBC MAC, FSE 2003.*
- NIST standard recommends using $t \geq 64$.