Radbound University

Disk Encryption and Message Authentication

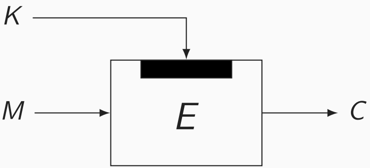
Applied Cryptography – Spring 2024

Bart Mennink

January 31, 2024

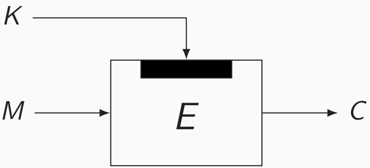
Institute for Computing and Information Sciences
Radboud University

Recap: Block Ciphers



- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size

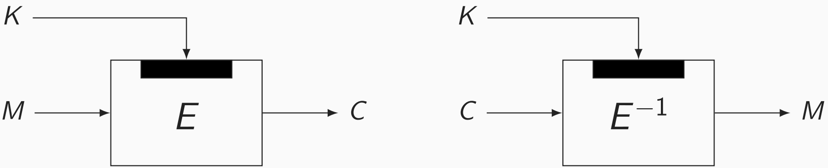
Recap: Block Ciphers



- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size
- Example [DR01]:

$$\begin{aligned}\text{AES-128}\colon \{0,1\}^{128} \times \{0,1\}^{128} &\to \{0,1\}^{128} \\ (K, M) &\mapsto C\end{aligned}$$

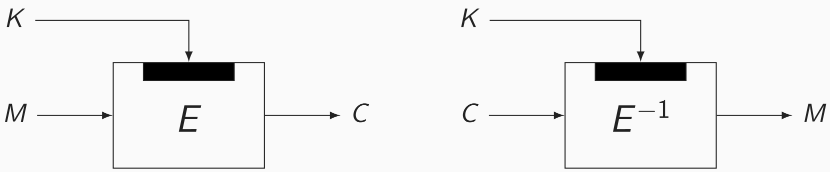
Recap: Block Ciphers



- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size
- Example [DR01]:

$$\text{AES-128}\colon \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$$
$$(K, M) \mapsto C$$

- For fixed key, E_K is invertible and the inverse is denoted as E_K^{-1}

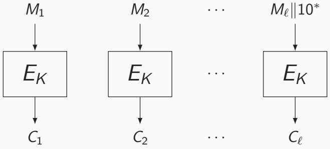


- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size
- Example [DR01]:

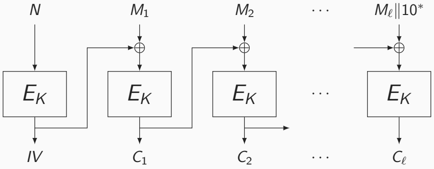
$$\text{AES-128}\colon \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$$
$$(K, M) \mapsto C$$

- For fixed key, E_K is invertible and the inverse is denoted as E_K^{-1}
- A good block cipher should behave like a random permutation

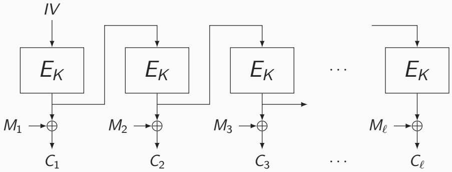
Block Cipher Encryption Modes



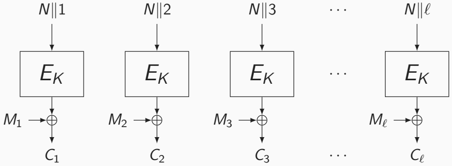
ECB



CBC



OFB



CTR

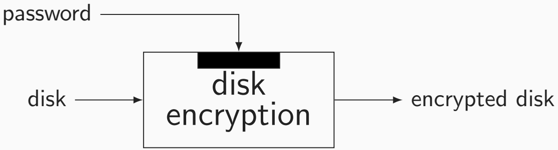
Last Lecture

- Keyed cryptographic constructions are modular:
  - A small primitive is turned into a larger mode of use
  - Typically, however, we only know how to build primitives that behave like random permutations
  - Most notable, block ciphers like AES
    - Mostly historically, but people still use them a lot!
    - E.g., each website over HTTPS sets up a TLS connection and reportedly over 70% over these connections use AES-GCM
- In this lecture:
  - Disk encryption
  - Message authentication
  - Beginning of authenticated encryption

Outline

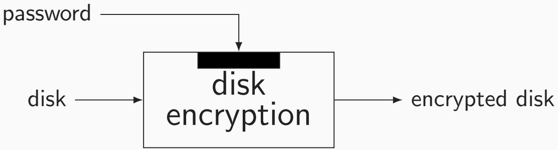
Disk Encryption

Disk Encryption



Main Goals

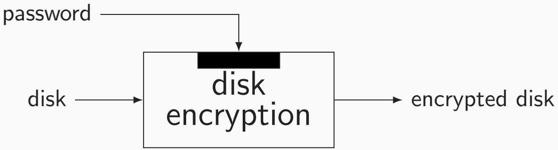
Disk Encryption



Main Goals

- Confidentiality
- Efficient in encryption and decryption
- No ciphertext expansion
- User friendly
- Incrementality . . . but not too much

Disk Encryption



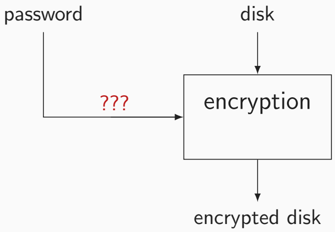
Main Goals

- Confidentiality
- Efficient in encryption and decryption
- No ciphertext expansion
- User friendly
- Incrementality . . . but not too much

These Slides

- High-level idea
- Core behind VeraCrypt,
  open source disk encryption tool

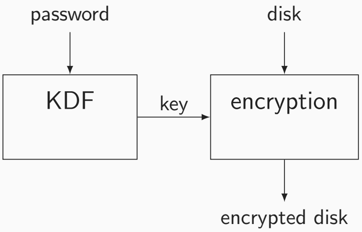
VeraCrypt Disk Encryption (1/2)



Encryption Scheme

- Actually encrypts all your data
- Requires some symmetric key . . . but we only have our password???

VeraCrypt Disk Encryption (1/2)



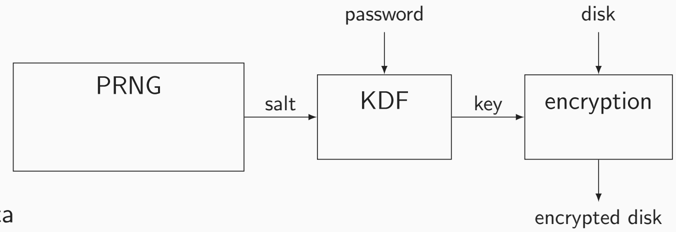
Encryption Scheme

- Actually encrypts all your data
- Requires some symmetric key ... but we only have our password???

KDF – Key Derivation Function

- Can “generate” a key from “limited information”
- But is this secure?

VeraCrypt Disk Encryption (1/2)



Encryption Scheme

- Actually encrypts all your data
- Requires some symmetric key . . . but we only have our password???

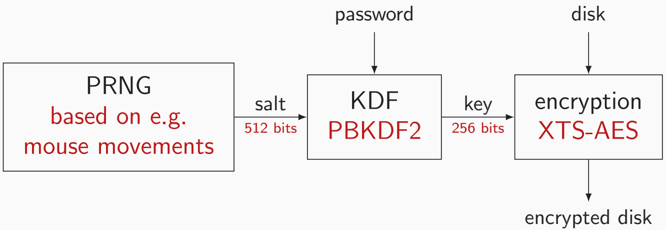
KDF – Key Derivation Function

- Can "generate" a key from "limited information"
- But is this secure?

PRNG – Pseudorandom Number Generator

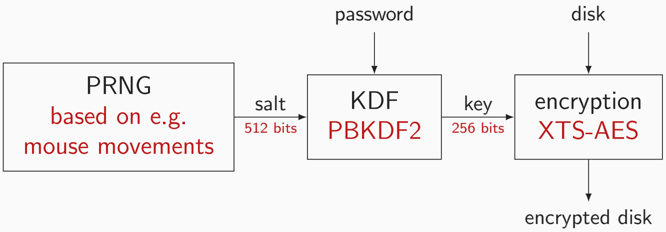
- Accumulates entropy from, e.g., mouse movements
- Turns it into a random looking string, the salt
- Salt fed to KDF to prevent dictionary attacks

VeraCrypt Disk Encryption (2/2)



In VeraCrypt

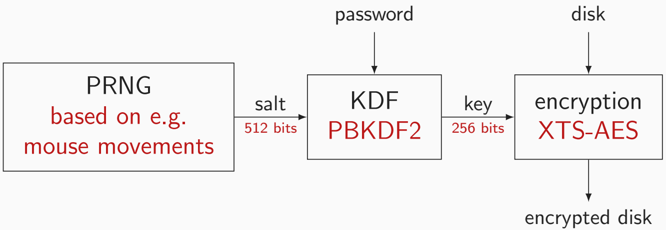
- Encryption scheme
  - XTS with AES-128 (more in Lecture 3)
  - Widely used standard for data encryption
- KDF
  - PBKDF2, hoped to behave like a pseudorandom function (more in Lecture 4)
  - Salt is 512 bits, so input has quite some “randomness”



In VeraCrypt

- Encryption scheme
  - XTS with AES-128 (more in Lecture 3)
  - Widely used standard for data encryption
- KDF
  - PBKDF2, hoped to behave like a pseudorandom function (more in Lecture 4)
  - Salt is 512 bits, so input has quite some “randomness”
- Weakest spot?

VeraCrypt Disk Encryption (2/2)

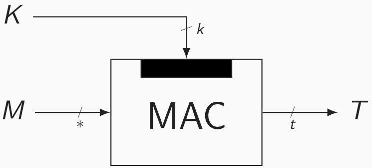


In VeraCrypt

- Encryption scheme
  - XTS with AES-128 (more in Lecture 3)
  - Widely used standard for data encryption
- KDF
  - PBKDF2, hoped to behave like a pseudorandom function (more in Lecture 4)
  - Salt is 512 bits, so input has quite some “randomness”
- Weakest spot?
  - The user! Short passwords are easy to guess
  - Salvaged by adjusting the number of rounds in PBKDF2

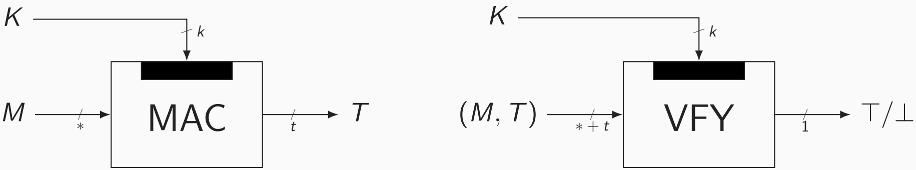
Message Authentication

Message Authentication (1/2)

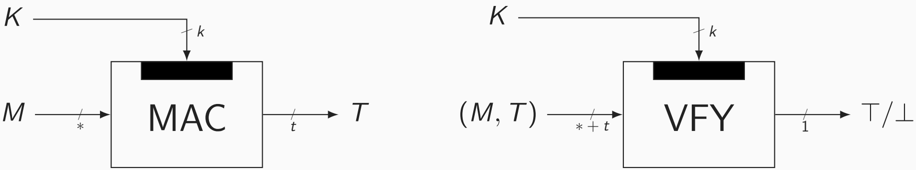


- Using key K, message M is signed with tag T

Message Authentication (1/2)

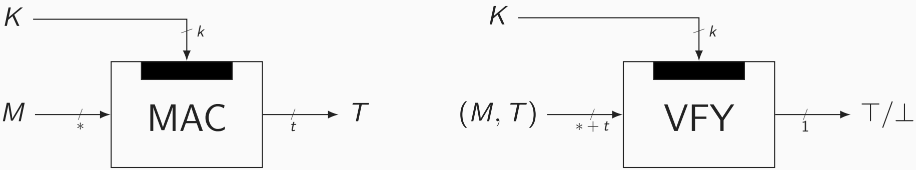


- Using key K, message M is signed with tag T
- Associated verification function takes K and (M, T) and outputs
  - $\top$ if tag is correct
  - $\perp$ if tag is incorrect



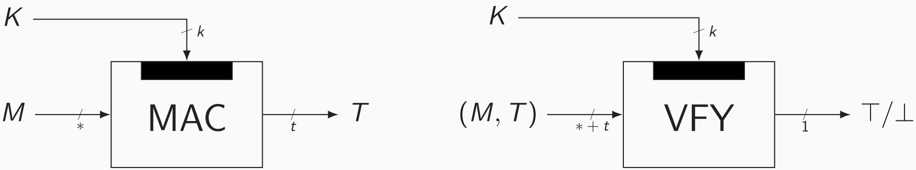
- Using key K, message M is signed with tag T
- Associated verification function takes K and (M, T) and outputs
  - $\top$ if tag is correct
  - $\perp$ if tag is incorrect
- Key and tag are typically of fixed size
- Message is typically of arbitrary length

Message Authentication (1/2)



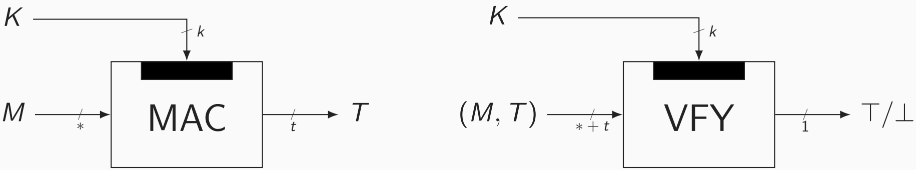
- Using key K, message M is signed with tag T
- Associated verification function takes K and (M, T) and outputs
  - $\top$ if tag is correct
  - $\bot$ if tag is incorrect
- Key and tag are typically of fixed size
- Message is typically of arbitrary length
- Sometimes, additional nonce: MAC-evaluations should be for unique nonce

Message Authentication (2/2)



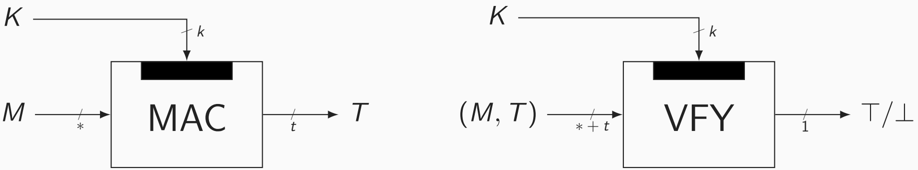
- Applications:
  - Message authentication: append tag to message
  - Entity authentication: compute tag over challenge

Message Authentication (2/2)



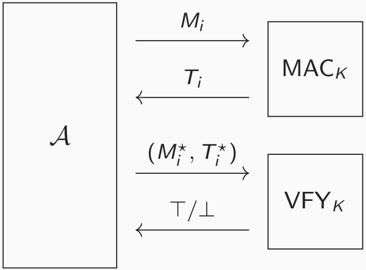
- Applications:
  - Message authentication: append tag to message
  - Entity authentication: compute tag over challenge
- Security goal: MAC_K should behave like a random function
  - I.e., $\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{MAC}}(q)$ should be small

Message Authentication (2/2)

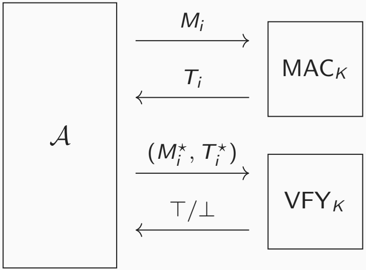


- Applications:
  - Message authentication: append tag to message
  - Entity authentication: compute tag over challenge
- Security goal: MAC_K should behave like a random function
  - I.e., $\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{MAC}}(q)$ should be small
- Often, one adopts a weaker notion, called unforgeability

- MAC security defined using unforgeability

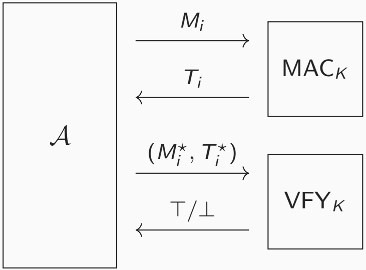


- MAC security defined using unforgeability
- Adversary $\mathcal{A}$ has access to MAC_K and VFY_K



Message Authentication Security

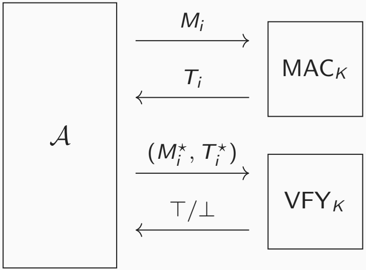
- MAC security defined using unforgeability
- Adversary $\mathcal{A}$ has access to MAC_K and VFY_K
- It can query both oracles interchangeably
  - No query VFY_K for an output of MAC_K
  - Optional: no repeated N for MAC_K



Message Authentication Security

- MAC security defined using unforgeability
- Adversary $\mathcal{A}$ has access to MAC_K and VFY_K
- It can query both oracles interchangeably
    - No query VFY_K for an output of MAC_K
    - Optional: no repeated N for MAC_K
- $\mathcal{A}$ mounts a forgery if VFY_K ever outputs $\top$
- Its advantage is defined as:

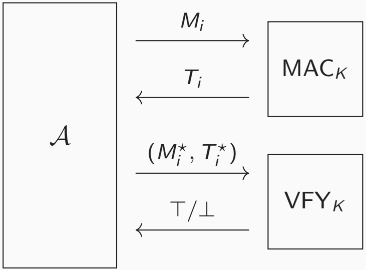
$$\mathbf{Adv}^{\mathrm{unf}}_{\mathsf{MAC}}(\mathcal{A}) = \mathbf{Pr}\left(\mathcal{A}^{\mathsf{MAC}_K,\mathsf{VFY}_K} \text{ forges}\right)$$



- MAC security defined using unforgeability
- Adversary $\mathcal{A}$ has access to MAC_K and VFY_K
- It can query both oracles interchangeably
  - No query VFY_K for an output of MAC_K
  - Optional: no repeated N for MAC_K
- $\mathcal{A}$ mounts a forgery if VFY_K ever outputs $\top$
- Its advantage is defined as:

$$\mathbf{Adv}^{\mathrm{unf}}_{\mathsf{MAC}}(\mathcal{A}) = \mathbf{Pr}\left(\mathcal{A}^{\mathsf{MAC}_K,\mathsf{VFY}_K} \text{ forges}\right)$$

- $\mathbf{Adv}^{\mathrm{unf}}_{\mathsf{MAC}}(q_m, q_v)$: supremal advantage over any $\mathcal{A}$ with:
  - query complexity q_m to MAC_K
  - query complexity q_v to VFY_K



Pseudorandomness Implies Unforgeability

Pseudorandomness Implies Unforgeability

- Consider a MAC function MAC_K
- Intuitively, if outputs of MAC_K look random, they should also be hard to forge
- Forging would then correspond to "guessing" a random output of MAC_K

Pseudorandomness Implies Unforgeability

- Consider a MAC function MAC_K
- Intuitively, if outputs of MAC_K look random, they should also be hard to forge
- Forging would then correspond to "guessing" a random output of MAC_K

- There is a well-established result that proves this:
  - Original result dates back to 1984, Goldreich et al. [GGM84]
  - In currently established formalism: Bellare et al. [BKR94,BGM04]

Pseudorandomness Implies Unforgeability

- Consider a MAC function MAC_K
- Intuitively, if outputs of MAC_K look random, they should also be hard to forge
- Forging would then correspond to "guessing" a random output of MAC_K

- There is a well-established result that proves this:
  - Original result dates back to 1984, Goldreich et al. [GGM84]
  - In currently established formalism: Bellare et al. [BKR94,BGM04]

$$\mathbf{Adv}^{\mathrm{unf}}_{\mathsf{MAC}}(q_m, q_v) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{MAC}}(q_m + q_v) + \frac{q_v}{2^t}$$

- Proof: see Theorem 6.2.2 of "Intro2Crypto-symmetric.pdf"

Universal Hash Functions

- Same interface as MAC, but weaker security requirements
  - Con: function should not be exposed to attacker
  - Pro: function can often be evaluated much cleanly

Universal Hash Functions

- Same interface as MAC, but weaker security requirements
  - Con: function should not be exposed to attacker
  - Pro: function can often be evaluated much cleanly
- Hash function family $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^t$ is called
  - δ-universal if $\mathbf{Pr}_K\left(H_K(M) = H_K(M')\right) \leq \delta \qquad (\forall\ M \neq M')$
  - ε-XOR-universal if $\mathbf{Pr}_K\left(H_K(M) \oplus H_K(M') = T\right) \leq \varepsilon \qquad (\forall\ M \neq M', T)$

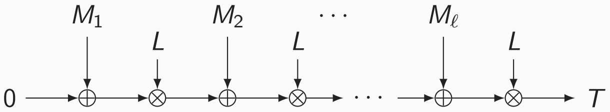
Universal Hashing

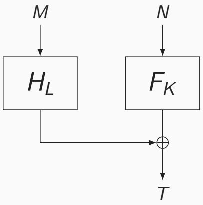
Universal Hash Functions

- Same interface as MAC, but weaker security requirements
  - Con: function should not be exposed to attacker
  - Pro: function can often be evaluated much cleanly
- Hash function family $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^t$ is called
  - δ-universal if $\mathbf{Pr}_K \left(H_K(M) = H_K(M') \right) \leq \delta \qquad (\forall\ M \neq M')$
  - ε-XOR-universal if $\mathbf{Pr}_K \left(H_K(M) \oplus H_K(M') = T \right) \leq \varepsilon \qquad (\forall\ M \neq M', T)$

GHASH

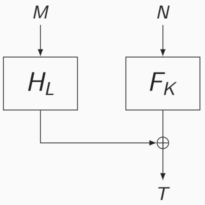
- Addition and multiplication over finite field
- $\ell 2^{-t}$-(XOR-)universal [MV04]





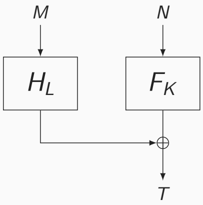
- Introduced in 1986 [WC81]
- Process arbitrary length M through universal hash, mask with $F_K(N)$

Wegman-Carter



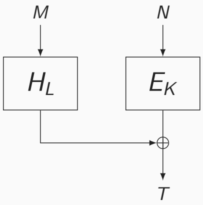
- Introduced in 1986 [WC81]
- Process arbitrary length M through universal hash, mask with $F_K(N)$
- Secure MAC function if
  - H is ε-XOR-universal
  - F is pseudorandom function
  - Nonce N never reused in MAC-queries

Wegman-Carter



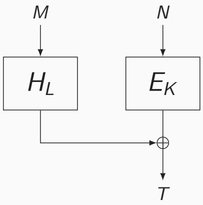
- Introduced in 1986 [WC81]
- Process arbitrary length M through universal hash, mask with $F_K(N)$
- Secure MAC function if
  - H is ε-XOR-universal
  - F is pseudorandom function
  - Nonce N never reused in MAC-queries
- $\mathbf{Adv}^{\mathrm{unf}}_{\mathrm{WC}}(q_m, q_v) \leq q_v\varepsilon + \mathbf{Adv}^{\mathrm{prf}}_{F}(q_m + q_v)$ [Ber05]

Wegman-Carter-Shoup

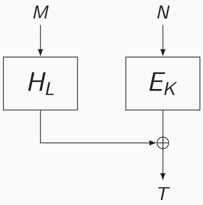


- PRFs hard to construct, PRPs easy: take a block cipher E
- Described by Shoup in 1996 [Sho96]

Wegman-Carter-Shoup



- PRFs hard to construct, PRPs easy: take a block cipher E
- Described by Shoup in 1996 [Sho96]
- Secure MAC function if
  - H is ε-XOR-universal
  - E is pseudorandom permutation
  - Nonce N never reused



- PRFs hard to construct, PRPs easy: take a block cipher E
- Described by Shoup in 1996 [Sho96]
- Secure MAC function if
  - H is ε-XOR-universal
  - E is pseudorandom permutation
  - Nonce N never reused
- $\mathbf{Adv}^{\mathrm{unf}}_{\mathsf{WC}}(q_m, q_v) \leq e^{(q_m+1)q_m/2^n} q_v \varepsilon + \mathbf{Adv}^{\mathrm{prp}}_{E}(q_m + q_v)$ [Ber05]

Poly1305 [Ber05]

- Wegman-Carter-Shoup construction with cleverly chosen universal hash
- Originally defined for AES, but commonly used with ChaCha20

Poly1305 [Ber05]

- Wegman-Carter-Shoup construction with cleverly chosen universal hash
- Originally defined for AES, but commonly used with ChaCha20

- Observation: 128-bit string X can be written as integer in $[0, 2^{128} - 1]$:

$$X[0] \| X[1] \| \cdots \| X[127] \in \{0,1\}^{128} \sim X[0] + 2X[1] + \cdots + 2^{127} X[127] \in [0, 2^{128} - 1]$$

Poly1305 [Ber05]

- Wegman-Carter-Shoup construction with cleverly chosen universal hash
- Originally defined for AES, but commonly used with ChaCha20

- Observation: 128-bit string X can be written as integer in $[0, 2^{128} - 1]$:

$$X[0] \| X[1] \| \cdots \| X[127] \in \{0,1\}^{128} \sim X[0] + 2X[1] + \cdots + 2^{127} X[127] \in [0, 2^{128} - 1]$$

- Keys $K, R \in \{0,1\}^{128}$ with 22 bits of R fixed
- On input of nonce $N \in \{0,1\}^{128}$ and message $M \in \{0,1\}^*$:

Poly1305 [Ber05]

- Wegman-Carter-Shoup construction with cleverly chosen universal hash
- Originally defined for AES, but commonly used with ChaCha20

- Observation: 128-bit string X can be written as integer in $[0, 2^{128} - 1]$:

$$X[0] \| X[1] \| \cdots \| X[127] \in \{0,1\}^{128} \sim X[0] + 2X[1] + \cdots + 2^{127} X[127] \in [0, 2^{128} - 1]$$

- Keys $K, R \in \{0,1\}^{128}$ with 22 bits of R fixed
- On input of nonce $N \in \{0,1\}^{128}$ and message $M \in \{0,1\}^*$:
    - M is partitioned into ℓ 128-bit blocks M_i

Poly1305 [Ber05]

- Wegman-Carter-Shoup construction with cleverly chosen universal hash
- Originally defined for AES, but commonly used with ChaCha20

- Observation: 128-bit string X can be written as integer in $[0, 2^{128} - 1]$:

$$X[0] \| X[1] \| \cdots \| X[127] \in \{0,1\}^{128} \sim X[0] + 2X[1] + \cdots + 2^{127} X[127] \in [0, 2^{128} - 1]$$

- Keys $K, R \in \{0,1\}^{128}$ with 22 bits of R fixed
- On input of nonce $N \in \{0,1\}^{128}$ and message $M \in \{0,1\}^*$:
  - M is partitioned into ℓ 128-bit blocks M_i
  - Each block (except last one) is appended with a 1
  - Call the updated blocks $C_1, \ldots, C_\ell$
  - Note, $C_1, \ldots, C_{\ell-1}$ are integers in $[2^{128}, 2^{129} - 1]$

Poly1305 [Ber05]

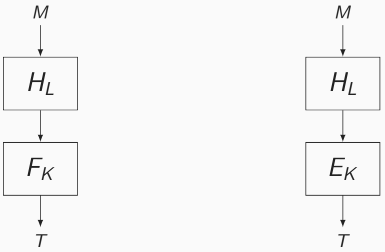
- Wegman-Carter-Shoup construction with cleverly chosen universal hash
- Originally defined for AES, but commonly used with ChaCha20

- Observation: 128-bit string X can be written as integer in $[0, 2^{128} - 1]$:

$$X[0] \| X[1] \| \cdots \| X[127] \in \{0,1\}^{128} \sim X[0] + 2X[1] + \cdots + 2^{127} X[127] \in [0, 2^{128} - 1]$$

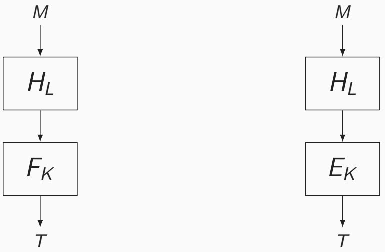
- Keys $K, R \in \{0,1\}^{128}$ with 22 bits of R fixed
- On input of nonce $N \in \{0,1\}^{128}$ and message $M \in \{0,1\}^*$:
  - M is partitioned into ℓ 128-bit blocks M_i
  - Each block (except last one) is appended with a 1
  - Call the updated blocks $C_1, \ldots, C_\ell$
  - Note, $C_1, \ldots, C_{\ell-1}$ are integers in $[2^{128}, 2^{129} - 1]$
  - $T = \Big(\big((C_1 \cdot R^\ell + C_2 \cdot R^{\ell-1} + \cdots + C_\ell \cdot R) \bmod 2^{130} - 5 \big) + E_K(N) \Big) \bmod 2^{128}$

Protected Hash (1/2)



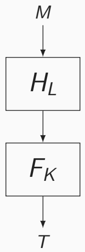
- Not a dedicated construction "as such", but appears quite frequently in disguise
  - CBC-MAC [BKR94]
  - Protected counter sum [Ber99]
- Process arbitrary length M through universal hash, protect with $F_K(\cdot)$ or $E_K(\cdot)$

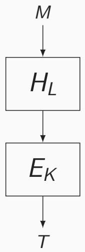
Protected Hash (2/2)



- Secure MAC function if
  - H is δ-universal
  - F is pseudorandom function

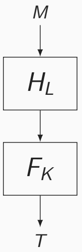
Protected Hash (2/2)

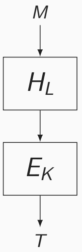




- Secure MAC function if
  - H is δ-universal
  - F is pseudorandom function
- $\mathbf{Adv}^{\mathrm{unf}}_{\mathsf{WC}}(q_m, q_v) \leq \binom{q_m+q_v}{2}\delta + q_v/2^t + \mathbf{Adv}^{\mathrm{prf}}_{F}(q_m + q_v)$ [Sho04]

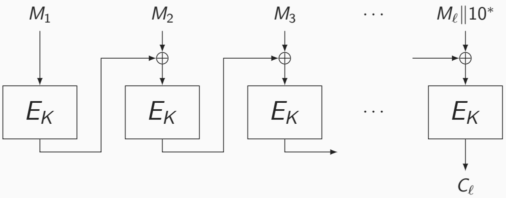
Protected Hash (2/2)





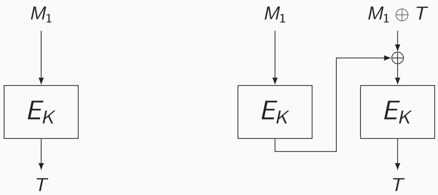
- Secure MAC function if
  - H is δ-universal
  - F is pseudorandom function
- $\mathbf{Adv}^{\mathrm{unf}}_{\mathrm{WC}}(q_m, q_v) \leq \binom{q_m+q_v}{2}\delta + q_v/2^t + \mathbf{Adv}^{\mathrm{prf}}_{F}(q_m + q_v)$ [Sho04]
- In case we protect with $E_K(N)$, extra loss of $\binom{q_m+q_v}{2}/2^n$

Cipher Block Chaining MAC (CBC-MAC) Mode



- In CBC encryption: C_i depends on $M_1, \ldots, M_i$
- Idea for message authentication:
  - Apply CBC with $IV = 0$ to padded message M
  - Define tag T to be the last ciphertext block
  - Important: discard all other ciphertext blocks!
- Turns out to be secure if messages are prefix-free

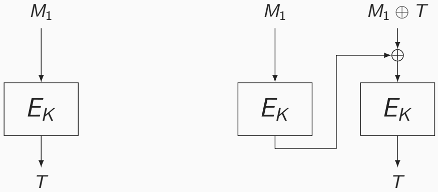
Cipher Block Chaining MAC (CBC-MAC) Mode: Weakness



- In general, CBC-MAC can be distinguished from random in two queries:
  - Query M_1, tag equals $T = E_K(M_1)$
  - Query $M_1 \| (M_1 \oplus T)$, tag equals

$$E_K(E_K(M_1) \oplus M_1 \oplus T) = E_K(T \oplus M_1 \oplus T) = E_K(M_1) = T$$

Cipher Block Chaining MAC (CBC-MAC) Mode: Weakness

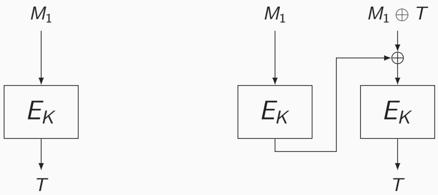


- In general, CBC-MAC can be distinguished from random in two queries:
  - Query M_1, tag equals $T = E_K(M_1)$
  - Query $M_1 \| (M_1 \oplus T)$, tag equals

$$E_K(E_K(M_1) \oplus M_1 \oplus T) = E_K(T \oplus M_1 \oplus T) = E_K(M_1) = T$$

- A random oracle would give independent responses
- CBC-MAC is not unforgeable and definitely not PRF-secure

Cipher Block Chaining MAC (CBC-MAC) Mode: Weakness

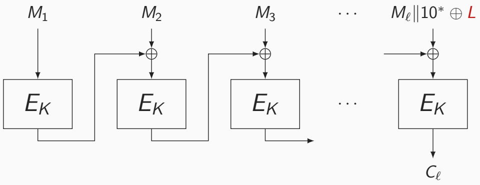


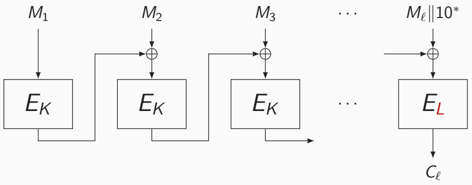
- In general, CBC-MAC can be distinguished from random in two queries:
  - Query M_1, tag equals $T = E_K(M_1)$
  - Query $M_1 \| (M_1 \oplus T)$, tag equals

$$E_K(E_K(M_1) \oplus M_1 \oplus T) = E_K(T \oplus M_1 \oplus T) = E_K(M_1) = T$$

- A random oracle would give independent responses
- CBC-MAC is not unforgeable and definitely not PRF-secure
- Note: attack ignores padding, but this can be dealt with

Cipher Block Chaining MAC (CBC-MAC) Mode: Fix

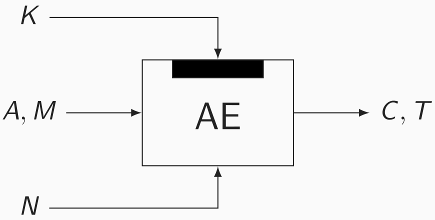




- Length-extension attack can be resolved by "special" finalization
- Solution 1: mask last block with dedicated key L (known as C-MAC)
- Solution 2: apply independent last primitive call
    - Can be seen as protected hash construction
- Both constructions indistinguishable from RO up to around $\binom{q}{2}/2^n$

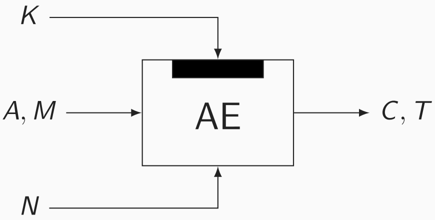
Authenticated Encryption (Teaser)

Authenticated Encryption



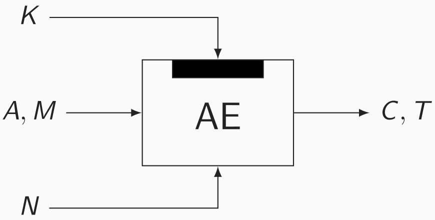
- Using key K:
  - Message M is encrypted in ciphertext C
  - Associated data A and message M are authenticated using T

Authenticated Encryption



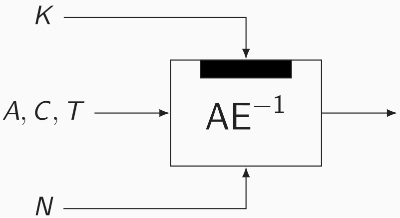
- Using key K:
  - Message M is encrypted in ciphertext C
  - Associated data A and message M are authenticated using T
- Nonce N randomizes the scheme

Authenticated Encryption



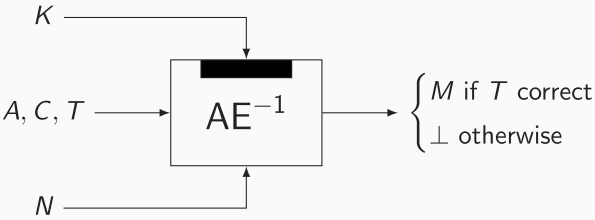
- Using key K:
  - Message M is encrypted in ciphertext C
  - Associated data A and message M are authenticated using T
- Nonce N randomizes the scheme
- Key, nonce, and tag are typically of fixed size
- Associated data, message, and ciphertext could be arbitrary length

Authenticated Decryption



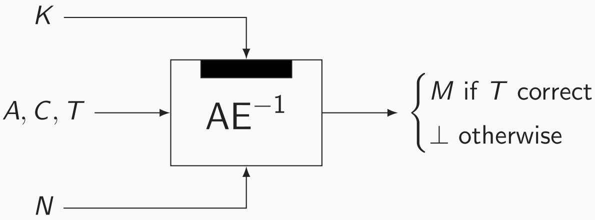
- Authenticated decryption needs to satisfy that
  - Message disclosed if tag is correct
  - Message is not leaked if tag is incorrect

Authenticated Decryption



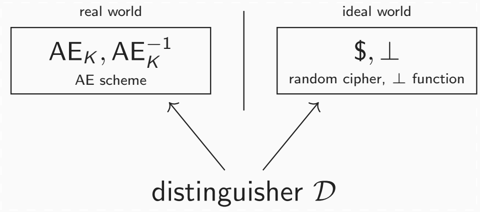
- Authenticated decryption needs to satisfy that
  - Message disclosed if tag is correct
  - Message is not leaked if tag is incorrect

Authenticated Decryption



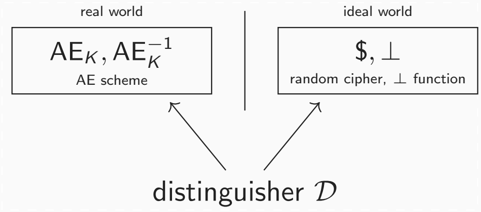
- Authenticated decryption needs to satisfy that
  - Message disclosed if tag is correct
  - Message is not leaked if tag is incorrect
- Correctness: $\mathrm{AE}_K^{-1}(N, A, \mathrm{AE}_K(N, A, M)) = M$

Authenticated Encryption Security



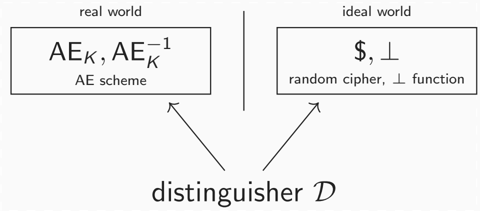
- Two oracles: (AE_K, AE_K^{-1}) (for secret key K) and $(\$, \perp)$ (secret)

Authenticated Encryption Security



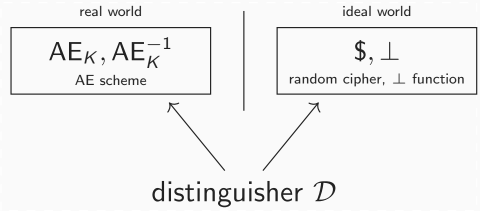
- Two oracles: (AE_K, AE_K^{-1}) (for secret key K) and $(\$, \perp)$ (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
  $\rightarrow$ unique nonce for each encryption query, and no trivial queries

Authenticated Encryption Security



- Two oracles: $(\mathrm{AE}_K, \mathrm{AE}_K^{-1})$ (for secret key K) and $(\$, \perp)$ (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
  $\rightarrow$ unique nonce for each encryption query, and no trivial queries
- $\mathcal{D}$ tries to determine which oracle it communicates with

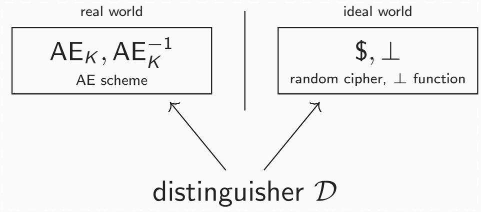
Authenticated Encryption Security



- Two oracles: $(\mathsf{AE}_K, \mathsf{AE}_K^{-1})$ (for secret key K) and $(\$, \perp)$ (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
  $\to$ unique nonce for each encryption query, and no trivial queries
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

$$\mathbf{Adv}_{\mathsf{AE}}^{\mathrm{ae}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(\mathsf{AE}_K, \mathsf{AE}_K^{-1} \; ; \; \$, \perp\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{AE}_K, \mathsf{AE}_K^{-1}} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\$, \perp} = 1\right)\right|$$

Authenticated Encryption Security



- Two oracles: $(\mathsf{AE}_K, \mathsf{AE}_K^{-1})$ (for secret key K) and $(\$, \perp)$ (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
  $\rightarrow$ unique nonce for each encryption query, and no trivial queries
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

$$\mathbf{Adv}^{\mathrm{ae}}_{\mathsf{AE}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(\mathsf{AE}_K, \mathsf{AE}_K^{-1}\ ;\ \$, \perp\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{AE}_K, \mathsf{AE}_K^{-1}} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\$,\perp} = 1\right)\right|$$

- $\mathbf{Adv}^{\mathrm{ae}}_{\mathsf{AE}}(q_e, q_v)$: supremal advantage over any $\mathcal{D}$ with query complexity q_e, q_v

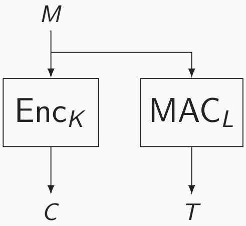
Generic Composition

- Generic constructions for AE:
  - Enc + MAC = AE

Generic Composition

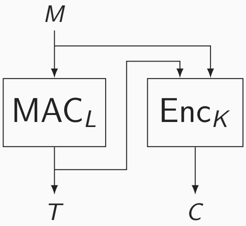
- Generic constructions for AE:
  - Enc + MAC = AE
- Bellare and Namprempre (2000): 3 basic approaches (N to both Enc/MAC, A to MAC)

E&M



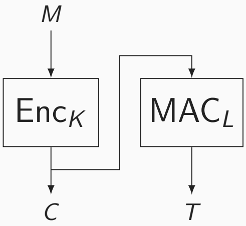
- Used in SSH

MtE



- Used in TLS

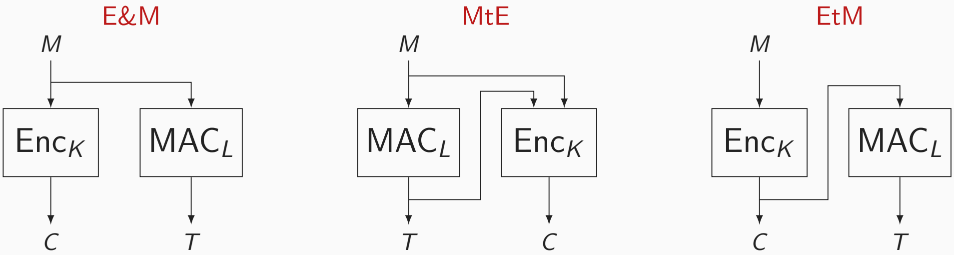
EtM



- Used in IPSec

Generic Composition

- Generic constructions for AE:
  - Enc + MAC = AE
- Bellare and Namprempre (2000): 3 basic approaches (N to both Enc/MAC, A to MAC)



- Used in SSH
- Generically insecure
  - $\mathrm{MAC}_L(M) = M \| T$?

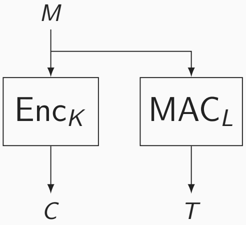
- Used in TLS

- Used in IPSec

Generic Composition

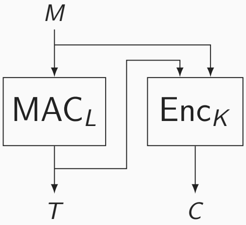
- Generic constructions for AE:
  - Enc + MAC = AE
- Bellare and Namprempre (2000): 3 basic approaches (N to both Enc/MAC, A to MAC)

E&M



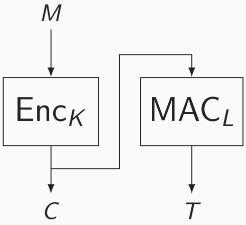
- Used in SSH
- Generically insecure
  - $\mathrm{MAC}_L(M) = M \| T$?

MtE



- Used in TLS
- Mildly insecure
- Padding oracle attack

EtM

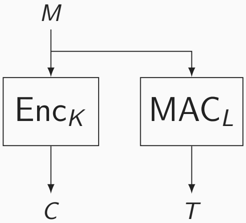


- Used in IPSec

Generic Composition

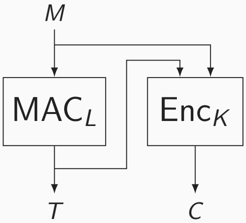
- Generic constructions for AE:
  - Enc $+$ MAC $=$ AE
- Bellare and Namprempre (2000): 3 basic approaches (N to both Enc/MAC, A to MAC)

E&M



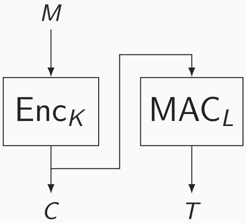
- Used in SSH
- Generically insecure
  - $\mathrm{MAC}_L(M) = M \| T$?

MtE



- Used in TLS
- Mildly insecure
- Padding oracle attack

EtM



- Used in IPSec
- Most secure variant
- Ciphertext integrity