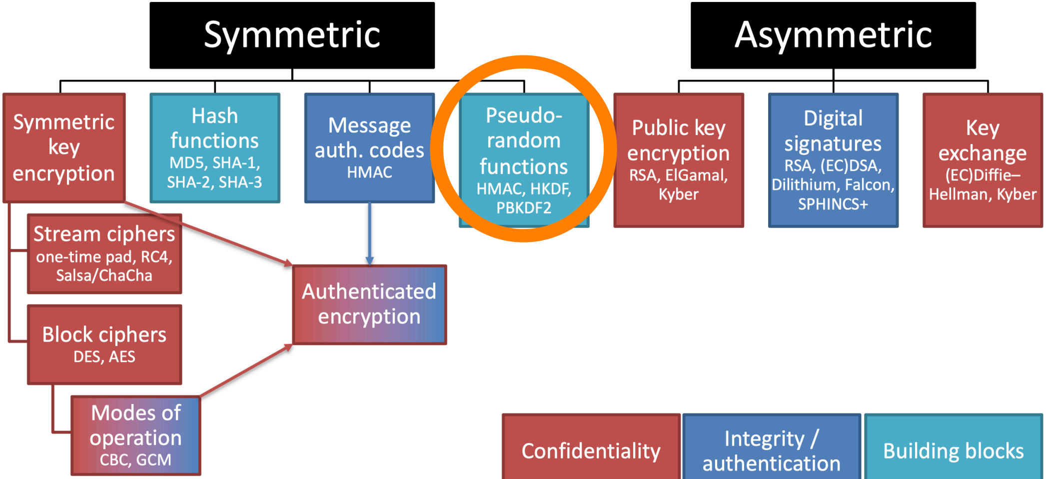
Topic 2.3
Integrity – Pseudorandom functions

Douglas Stebila

CO 487/687: Applied Cryptography

Fall 2024

Map of cryptographic primitives



Outline

Key derivation functions and pseudorandom functions

Pseudorandom generators

Definition (Pseudorandom generator)

A pseudorandom generator is a deterministic function that takes as input a random seed $k \in \{0,1\}^\lambda$ and outputs a random-looking binary string of length ℓ.

$$\text{PRG} : \{0,1\}^\lambda \to \{0,1\}^\ell$$

Some definitions of PRGs allow for arbitrary output, like when using PRGs for keystream generators in stream ciphers.

Pseudorandom functions

Definition (Pseudorandom function)

A pseudorandom function is a deterministic function that takes as input a random seed $k \in \{0,1\}^\lambda$ and a (non-secret) label in $\{0,1\}^*$ and outputs a random-looking binary string of length ℓ.

$$\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\ell$$

Security property for PRGs and PRFs

Indistinguishability: Assuming the seed is uniformly random on $\{0, 1\}^\lambda$, it should be computationally infeasible for an adversary to distinguish the output of a PRG/PRF from a uniformly random string.

For PRGs, the adversary gets either the real output of the PRG under an unknown seed, or a random output, and must decide which.

For PRFs, the adversary can make many calls to an oracle where the adversary can supply a label, and either always gets the real output of the PRF using the same unknown seed applied to the label, or always gets a randomly chosen output (for distinct labels), and must decide which.

PRGs and PRFs assume that the random seed is a truly random (uniform) secret.

What if the seed is high entropy but non-uniform?

Key derivation functions

Definition (Key derivation function)

A key derivation function is a deterministic function that takes as input a random seed $k \in \{0,1\}^\lambda$ and a (non-secret) label in $\{0,1\}^*$ and outputs a random-looking binary string of length ℓ.

$$\mathsf{KDF} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\ell$$

Difference between KDFs and PRFs: KDF output should be indistinguishable from random even if the key k is non-uniform but sufficiently high entropy.

Uses of PRGs, PRFs, KDFs

PRGs: expanding a strong short key into a long key (e.g., stream cipher)

PRFs: deriving many keys from a single key

- If Alice and Bob share one key k, but they need two keys for their application (e.g., one for encryption, one for MAC), they can compute $k' = \text{PRF}(k, \text{“label for encryption”})$, $k'' = \text{PRF}(k, \text{“label for MAC”})$

KDFs: turning longer non-uniform keys into shorter uniform keys

HMAC as a PRG/PRF/KDF

PRG: compute $\mathsf{HMAC}_k(1) \| \mathsf{HMAC}_k(2) \| \mathsf{HMAC}_k(3) \| \ldots$

PRF: compute $\mathsf{HMAC}_k(label)$

KDF: compute $\mathsf{HMAC}_{label}(k)$

Better way: special construction HKDF based on HMAC which first “extracts” high entropy secret from the keying material (similar to a KDF) then “expands” it to arbitrary length output with a label (like a PRF).

Application: Key stretching

Goal

Convert a (possibly) short and weak key into a long(er) and (more) secure key.

Useful for:

- Password storage in databases (harder to brute-force)
- Fixed-length keys (user types in a passphrase which is converted into a 256-bit key)
- Key exchange using public-key cryptography

Key stretching techniques

- Bad:
  - Secret prefix method using an iterated hash function.
  - Secret suffix method using an iterated hash function.
- Better:
  - HMAC using an iterated hash function.
  - Secret prefix method, using a (non-iterated) hash function specifically designed to support this usage (e.g. SHA-3)
- Best:
  - Key derivation function, such as PBKDF2, `scrypt`, `bcrypt`, Argon2.
  - Krawczyk, 2010: security definitions and proofs for KDFs

Password-Based Key Derivation Function 2 (PBKDF2)

$$k = \mathsf{PBKDF2}(F, p, s, c, \ell)$$

where

$$F = \text{keyed hash function}$$
$$p = \text{passphrase}$$
$$s = \text{salt}$$
$$c = \text{number of iterations}$$
$$\ell = \text{output length}$$

- This function is **supposed to be slow**. Larger iteration counts yield more security (and slower performance).
- Standardization: PKCS #5 v2.0, RFC 2898, NIST SP800-132.

PBKDF2

Recall $c =$ number of iterations, and $\ell =$ output length.

$$\text{PBKDF2}(F, p, s, c, \ell) = T_1 \| T_2 \| \cdots \| T_\ell$$

where

$$\begin{aligned} T_i &= U_{i,1} \oplus U_{i,2} \oplus \cdots \oplus U_{i,c} \\ U_{i,1} &= F(p, s \| i) \\ U_{i,2} &= F(p, U_{i,1}) \\ &\vdots \\ U_{i,c} &= F(p, U_{i,c-1}) \end{aligned}$$

Each T_i requires c calls to F. Slow down computation by choosing large c.

PBKDF2

Example

WPA2 (Wi-Fi Protected Access) uses the following function to derive a 256-bit key (truncated from 320 bits) from a passphrase.

$$k = \mathsf{PBKDF2}(\mathsf{HMAC\text{-}SHA1}, \mathsf{passphrase}, \mathsf{ssid}, 4096, 2)$$

Example

iOS 4 used PBKDF2 with $c = 10000$. LastPass used $c \geq 100000$.

Modern alternatives to PBKDF2: bcrypt, scrypt, Argon2.

4+8+7 things to remember from CO 487

Symmetric key primitives

Pseudorandom functions

$F(k, \text{label}) \rightarrow x$

- Security goal: without secret key, output looks indistinguishable from random.
- Related primitives: pseudorandom generators, key derivation functions.
- Secure options as of 2024:
  - HMAC-SHA256 (or better), HKDF
  - For hashing passwords, use intentionally slow functions PBKDF2 or Argon2 with a random salt to slow down brute force searches.