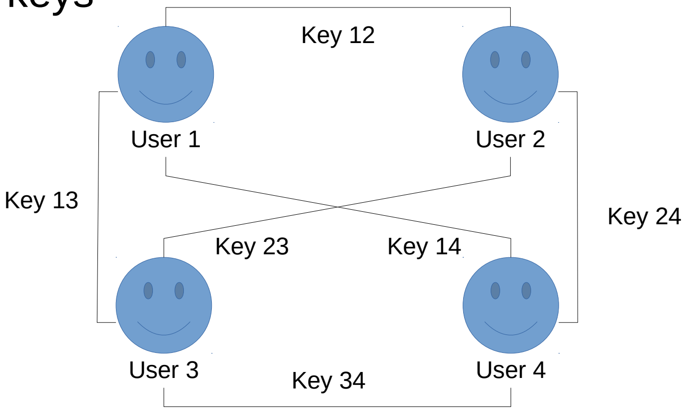
Key Exchange

Contents

Key management

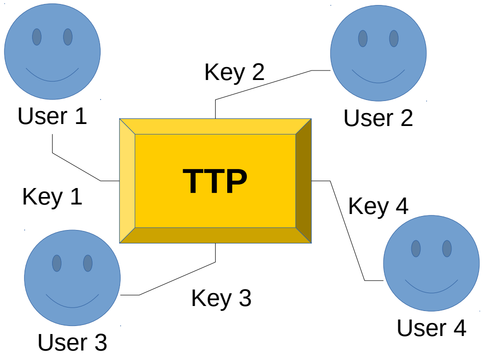
- Storing mutual secret keys is difficult
- In a universe of *n* users, each user requires *O(n)* keys



Adaptation of: Dan Boneh, Cryptography I, Stanford.

On-line Trusted Third Party (TTP)

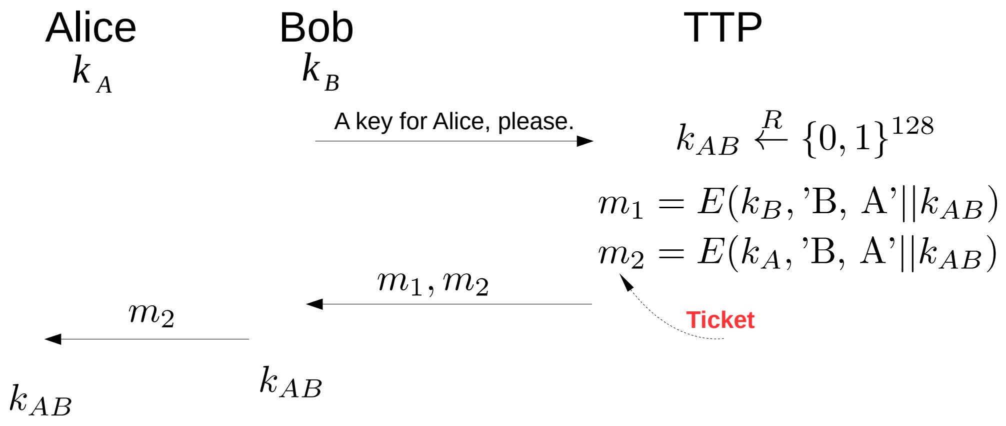
- Every user has to manage only **a single key**
  - The one used to communicate with TTP
- Upon request, the TTP generates shared secret keys for user sessions



Adaptation of: Dan Boneh, Cryptography I, Stanford.

TTP: Generating keys (toy protocol)

- Bob wants a shared secret with Alice



Alice k_A

Bob k_B

TTP

A key for Alice, please.

$k_{AB} \xleftarrow{R} \{0,1\}^{128}$

$m_1 = E(k_B, \text{'B, A'} || k_{AB})$

$m_2 = E(k_A, \text{'B, A'} || k_{AB})$

Ticket

m_1, m_2

m_2

k_{AB}

k_{AB}

(E, D) a CCA secure cipher.

TTP: Security

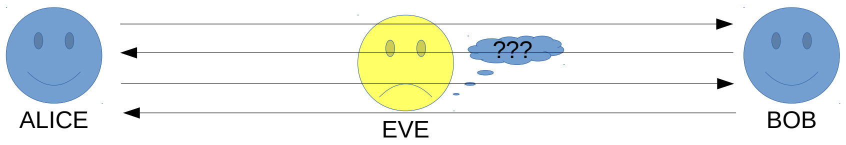
- An eavesdropper sees
  - $m_1 = E(k_B, \text{'B, A'} || k_{AB})$
  - $m_2 = E(k_A, \text{'B, A'} || k_{AB})$
- Since (E, D) is CCA secure, she learns nothing about k_{AB}
- Issues
  - TTP needed for all key exchanges
  - TTP knows all user and all session keys
  - Replay attacks possible
- Basis of Kerberos

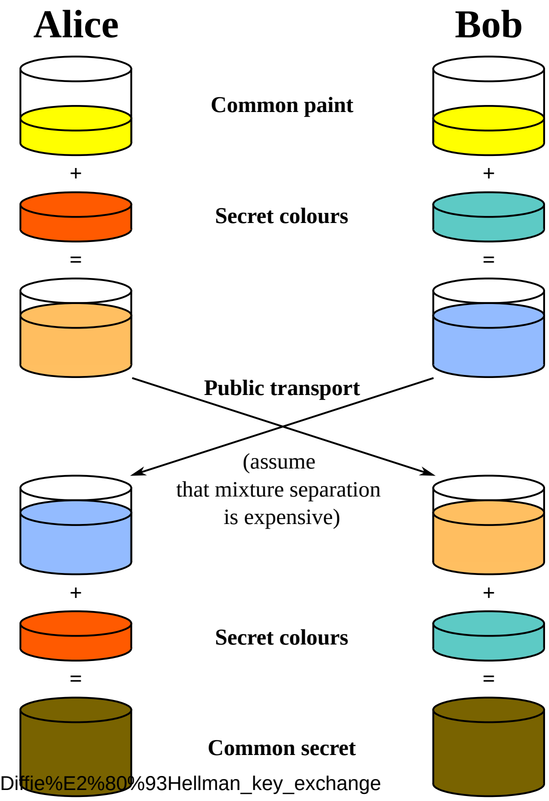
The main issue

- Can we generate shared keys without an **on-line** TTP?
  - YES!
- Entrance of public-key cryptography
- Two most widely known constructions
  - Diffie-Hellman protocol (1976)
  - RSA crypto system (1977)

Diffie-Hellman protocol

- Stems from hard problems in algebra
- Alice an Bob want to establish a shared secret in the presence of an eavesdropper
- Security against eavesdropping only



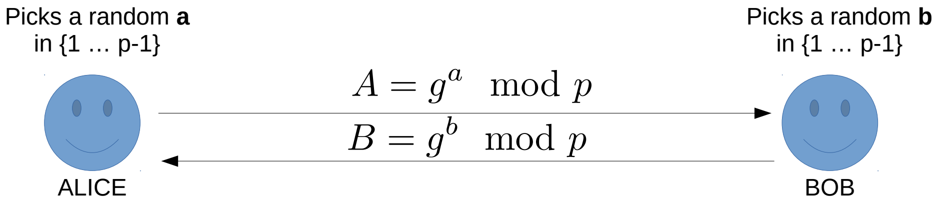


https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

---

Diffie-Hellman protocol (informally)

- Fix a large prime **p** (600 digits ~ 2kbits long)
- Fix an integer **g** in ***G*** = {1 … p-1} such that **g** is a primitive root modulo **p** (generator)
  - Raising **g** to powers of 0 to p-2 generates all values in {1 … p-1}



$$A = g^a \mod p$$

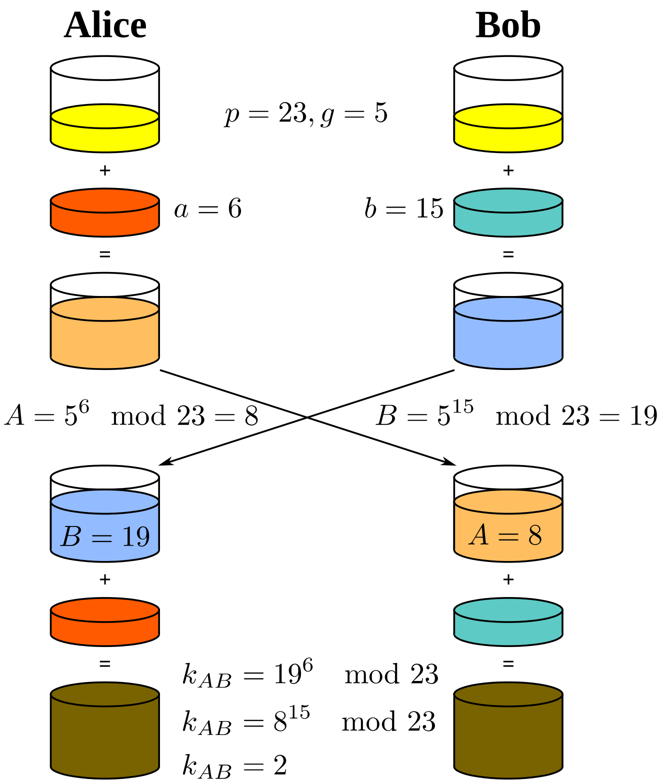
$$B = g^b \mod p$$

$$k_{AB} = g^{ab} \mod p$$

$B^a = (g^b)^a = g^{ab} \mod p$ $\qquad$ $A^b = (g^a)^b = g^{ab} \mod p$

Adaptation of: Dan Boneh, Cryptography I, Stanford.

$5^0 = 1 \mod 23$
$5^1 = 5 \mod 23$
$5^2 = 2 \mod 23$
$5^3 = 10 \mod 23$
$5^4 = 4 \mod 23$
$5^5 = 20 \mod 23$
$5^6 = 8 \mod 23$
$5^7 = 17 \mod 23$
$5^8 = 16 \mod 23$
$5^9 = 11 \mod 23$
$5^{10} = 9 \mod 23$
$5^{11} = 22 \mod 23$
$5^{12} = 18 \mod 23$
$5^{13} = 21 \mod 23$
$5^{14} = 13 \mod 23$
$5^{15} = 19 \mod 23$
$5^{16} = 3 \mod 23$
$5^{17} = 15 \mod 23$
$5^{18} = 6 \mod 23$
$5^{19} = 7 \mod 23$
$5^{20} = 12 \mod 23$
$5^{21} = 14 \mod 23$
$5^{22} = 5^0 = 1 \mod 23$



$p = 23, g = 5$

$a = 6$ $\qquad$ $b = 15$

$A = 5^6 \mod 23 = 8$ $\qquad$ $B = 5^{15} \mod 23 = 19$

$B = 19$ $\qquad$ $A = 8$

$$k_{AB} = 19^6 \mod 23$$

$$k_{AB} = 8^{15} \mod 23$$

$$k_{AB} = 2$$

Security (informally)

- An eavesdropper sees
  - $p, g, A = g^a (mod\ p), B = g^b (mod\ p)$
- Can she derive $g^{ab} (mod\ p)$ herself?
- In general, let's define

$$\mathrm{DH}_g(g^a, g^b) = g^{ab} \qquad (mod\ p)$$

- How difficult is to compute DH function (mod p)?

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Security (informally)

- Suppose *p* is *n* bits long
- Best known algorithm (GNFS) computes function DH in $e^{O(\sqrt[3]{n})}$
- How difficult is to break DH compared to breaking a symmetric cipher?

Cipher key size	DH modulus size [in modulo primes]	DH modulus size [Elliptic Curve]
80	1024	160
128	3072	256
256	15360	512

- Slow transition from (mod *p*) to elliptic curves

Adaptation of: Dan Boneh, Cryptography I, Stanford.

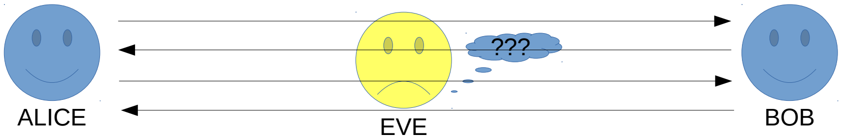
DH: Open issues

- Remember: security against eavesdropping only
- An active attacker can break the protocol with the man-in-the-middle attack
  - Reason: exchanges are not authenticated

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Public key encryption for key exchange

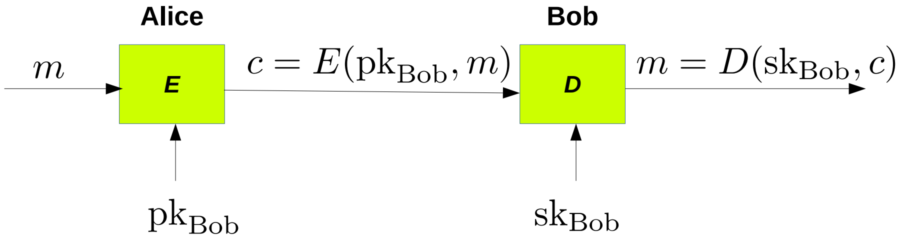
- Alice an Bob want to establish a shared secret in the presence of an eavesdropper
- Security against eavesdropping only



Adaptation of: Dan Boneh, Cryptography I, Stanford.

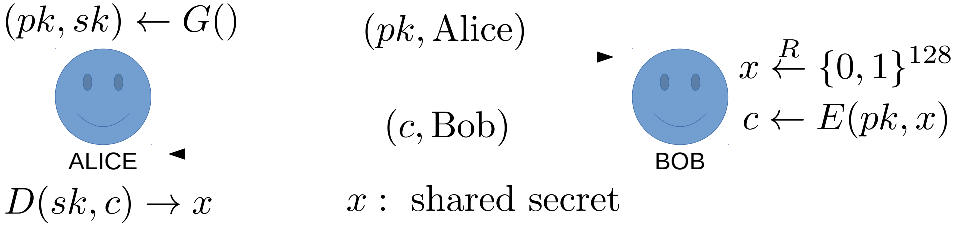
Public key encryption

- Each party uses a key pair: $k = (\text{pk}, \text{sk})$
- Public key is given to everyone, secret is kept hidden



Adaptation of: Dan Boneh, Cryptography I, Stanford.

Establishing a shared secret



$(pk, sk) \leftarrow G()$

(pk, Alice)

$x \xleftarrow{R} \{0,1\}^{128}$

$c \leftarrow E(pk, x)$

(c, Bob)

ALICE

BOB

$D(sk, c) \rightarrow x$

$x:$ shared secret

- Adversary sees $pk, \; E(pk, x)$
- Adversary wants x
- If $\zeta = (G, E, D)$ is sem. secure, the adv. obtains no information about x
- Security **against eavesdropping only**: protocol still vulnerable to man-in-the-middle

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Digital signatures

- Preserving integrity in public-key cryptography
  - "MACs" of public-key cryptography
- Idea: The signer signs a message with her secret key. Anyone can verify the signature using the corresponding public key and thus know:
  - That the message has not been tampered with
  - That the signer indeed signed the message
- Similar to MACs, but digital signatures are
  - **Publicly verifiable**: anyone (with PK) can verify the signature
  - **Non-repudiative**: the signer cannot later deny having signed a particular message

Signature scheme: def.

- **Def:** A signature scheme (G, S, V) is a triple of eff. algs. defined over (M, Z) where:
  - $G()$ is a rand. alg. that generates key pairs (pk, sk)
  - $S(sk, m)$ is an alg. that signs a message $m \in M$ using secret key sk and produces a signature $z \in Z$
  - $V(pk, m, z)$ is a det. alg. that verifies the signature $z \in Z$ of message $m \in M$ using pk and outputs **1** if the signature verifies, or **0** otherwise
  - A signature generated by *S* must always verify by *V*:

$$\forall (pk, sk), m \in M : \Pr[V(pk, m, S(sk, m)) = 1] = 1$$

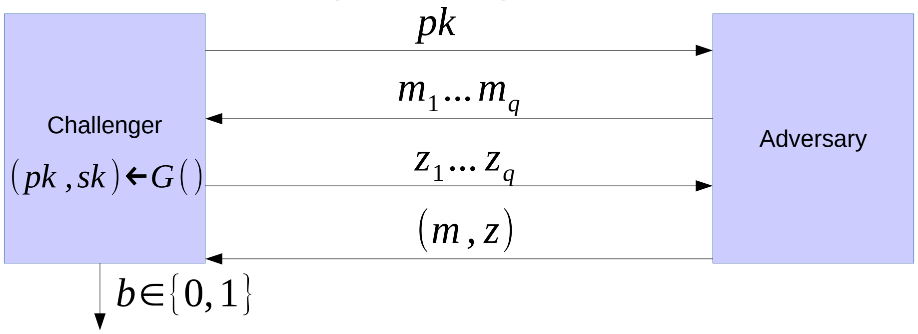
Digital signatures: Threat model

- Attacker's power: **Chosen message attack**
  - For $m_1 \ldots m_q$ attacker is given $z_i = S(sk, m_i)$
- Attacker's goal: **Existential forgery**
  - Produce a **new** valid (m, z) s. t.

$$m \notin \{m_1 \ldots m_q\}$$

→ *An adversary cannot produce a valid signature for a new message*

Secure digital signature: def.



Challenger

$(pk, sk) \leftarrow G()$

pk

$m_1 \ldots m_q$

$z_1 \ldots z_q$

(m, z)

Adversary

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

$$\begin{array}{ll} b=1 & \text{if } V(pk, m, z)=1 \text{ and } m \notin \{m_1 \ldots m_q\} \\ b=0 & \text{otherwise} \end{array}$$

A signature scheme (G, S, V) is **secure** if for all “efficient” adversaries A:

$$\mathrm{Adv}_{\mathrm{SIG}}[A, I] = \Pr[\text{Chal. outputs } 1]$$

is “negligible”.

Extending the message space

- **Hash-and-sign paradigm**
  - *Constructing a signature scheme for large messages from a signature scheme for small messages (and strengthening security)*
- **Thm.** Let (G, S, V) be a secure signature scheme over (M, Z) and let $H: M' \rightarrow M$ be a collision resistant hash function where $|M'| \gg |M|$. Then (G, S', V') is also secure sig. scheme, where:

$$S'(sk, m) := S(sk, H(m))$$

$$V'(pk, m, z) := V(pk, H(m), z)$$

Signatures from TDP: Full Domain Hash

- Building blocks
  - ***(G, F, F*** $^{-1}$ ***)*** – Secure trapdoor permutation (TDP)
    - ***F: X*** → ***X***
  - ***H: M*** → ***X*** – collision resistant hash function
- Full domain (length) hash (FDH)
  - ***G()*** from TDP
  - $S(sk, m) := F^{-1}(sk, H(m))$
  - $V(pk, m, z) := \begin{cases} 1 & H(m) = F(pk, z) \\ 0 & \text{otherwise} \end{cases}$

Signatures from TDP: Full Domain Hash

- **Thm.** Let ***(G, F, F*** $^{-1}$ ***)*** be a secure TDP ***X*** → ***X*** and let ***H: M*** → ***X*** be a collision resistant hash function. Then signature scheme FDH is secure if ***H*** is a *random oracle*.
- FDH produces <u>unique signatures</u>: every message has its own signature

Signatures from TDP: Full Domain Hash

- Hashing is required for security; schemes without hashing are insecure. For instance:

$$S(sk,m) := F^{-1}(sk,m) \qquad V(pk,m,z) := F(pk,m) == z$$

  - **Zero-message attack**: create an existential forgery by picking a random signature, and creating a "message" from it

$$z \overset{R}{\leftarrow} Z, m \leftarrow F(pk,z)$$

  - **Multiplicative-property attack** (when using RSA)
    - Ask for signatures on two messages $m_{1,} m_2$

$$z_1 \leftarrow S(sk,m_1), z_2 \leftarrow S(sk,m_2)$$

    - Output existential forgery

$$m_3 \leftarrow m_1 \cdot m_2$$
$$z_3 \leftarrow z_1 \cdot z_2$$

Signatures from RSA trapdoor

- ***G()***
  - Choose random primes p, q (~1024 bits); $N = p \cdot q$
  - Choose integers e, d such that $e \cdot d = 1 \mod \varphi(N)$
  - Return $pk = (N,e), \ sk = (N,d)$
- $S((N,d),m) := H(m)^d \mod N$
- $V((N,e),m,z) := \begin{cases} 1 & H(m) = z^e \mod N \\ 0 & \text{otherwise} \end{cases}$
- What about ***H***?

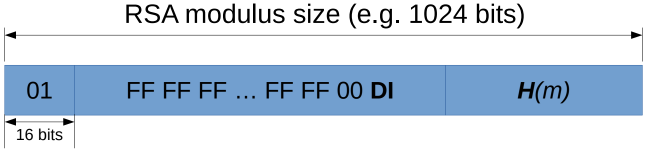
RSA Full Domain Hash

- We require ***H: M*** → $\mathbb{Z}_N^*$
  - The output length of ***H*** depends on *N*; could be different for every public key
  - Ideally we want the output length of ***H*** to be fixed
- **Thm.** Let ***H: M*** → ***Y*** be a collision resistant hash function where ***Y*** = *{1, …,* 2^{n-2}*}* and *n* is the number of bits used to represent *N*. Then RSA-FDH is secure sig. scheme if ***H*** is a random oracle.

→ The bit-length of digests must be of similar length as is the bit-length of the modulus $|Y| \geq N/4$

PKCS1 v1.5 signatures

- Widely deployed (TLS certificates, S/MIME, ...)



- **DI** – digest info encodes the name of the used hash function ***H*** (SHA*, MD*, …)
- The resulting value is then signed by raising it to *d* in *mod N* (recall, *sk = (N, d)*)
- Not FDH, but partial domain hash
  - No security proof; also no known substantial attacks
    - Issue with proving: ***H***(m) maps to a small subset of $\mathbb{Z}_N^*$

Probabilistic Signature Scheme (PSS)

- Randomizes the signature with a public random value *s* called **salt**
- $S((N,d),m,s) := [H(s||m)\ ||\ MGF[H(s||m)] \oplus s]^d \mod N$
  - MGF – mask generating function that extends the hash size to the full modulus size
- $V((N,e),m,z,s) := \begin{cases} 1 & H(s||m)\ ||\ MGF[H(s||m)] \oplus s = z^e \mod N \\ 0 & \text{otherwise} \end{cases}$
  - Provably secure in random oracle model
  - Part of PKCS1 v2.1

Adaptation of: Dan Boneh, Cryptography I, Stanford.

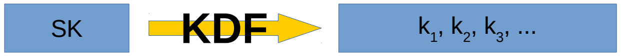
Digital Signature Standard (DSS)

- NIST (FIPS 186)
  - Also called Digital Signature Algorithm (**DSA**)
- Relies on the hardness of Dlog
- No known proof of security
  - But also no serious attacks found
- Has an equivalent in elliptic curves (**ECDSA**)

Adaptation of: Dan Boneh, Cryptography I, Stanford.

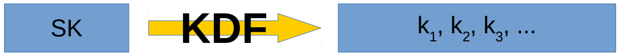
Deriving many keys from one

- **Scenario**: we obtain a single <u>source key</u> (SK)
  - From a hardware random number generator
  - From a key exchange protocol
- We need many keys to secure the session
  - Unidirectional keys, MAC/encryption keys
- **Goal**: generate many keys from a single SK
  - KDF – key derivation function



Deriving many keys from one

- Three cases
  1) SK is uniform in key space
  2) SK is non-uniform in key space
  3) SK is a password



Key derivation: (1) SK is uniform

- Let PRF F: K × X → $\{0, 1\}^n$
- If source key is **uniform** in K:

$$KDF(sk, ctx, l) := F(sk, ctx||0) \ || \ F(sk, ctx||1) \ || \ ... \ || \ F(sk, ctx||l)$$

- **ctx**: a string unique to every application
  - Assures that two applications derive independent keys even if they sample the same source key

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Key derivation: (2) SK is non-uniform

- The KDF can be directly used only when SK is uniform

  → If SK is not uniform, the PRF output may not look random

- Reasons for non-uniformity of SK
  - Hardware RNG may be *biased*
  - Key-exchange protocol may produce a key that is *uniform in some subset* of K

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Key derivation: (2) SK is non-uniform

- **Extract-then-Expand paradigm**
  - Step 1) Use an **extractor** and **SK** to extract a pseudo-random key **k** that is uniform in key space
    - Use **salt:** a fixed public (non-secret) random string
  - Step 2) expand ***k*** with KDF
- **HKDF** – a KDF from HMAC
  - Step 1) ***k*** ← HMAC(salt, SK)
  - Step 2) Expand as you would with uniform keys, but use HMAC for PRF and ***k*** for key
  - https://tools.ietf.org/html/rfc5869

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Key derivation: (3) SK is a password

- Particular care needed when deriving keys from passwords
  - HKDF unsuitable here: passwords have low entropy
  - Derived keys will be vulnerable to dictionary attack
- General idea: add **salt** and **slow down hashing**
- **PBKDF** – password-based KDF
  - PKCS #5 v2.0 and https://tools.ietf.org/html/rfc2898
  - Iterate hash function many times

Adaptation of: Dan Boneh, Cryptography I, Stanford.

Final words

- Cryptography is a powerful tool, but it is too easy to use it incorrectly
  - Systems work, but could be easily attacked
- To reduce the probability of making mistakes
  - Have others review your design and code
  - Never invent your own primitives (ciphers, MACs, modes of operation, …)
  - Avoid implementing your own cryptographic operations
    - E.g. instead of combining AES-CTR and HMAC, prefer AES-GCM

Adaptation of: Dan Boneh, Cryptography I, Stanford.