CIS 6930: IoT Security

Lecture 4

Prof. Kaushal Kafle

Spring 2025

Derived from slides by Adwait Nadkarni, William Enck, Micah Sherr and Patrick McDaniel

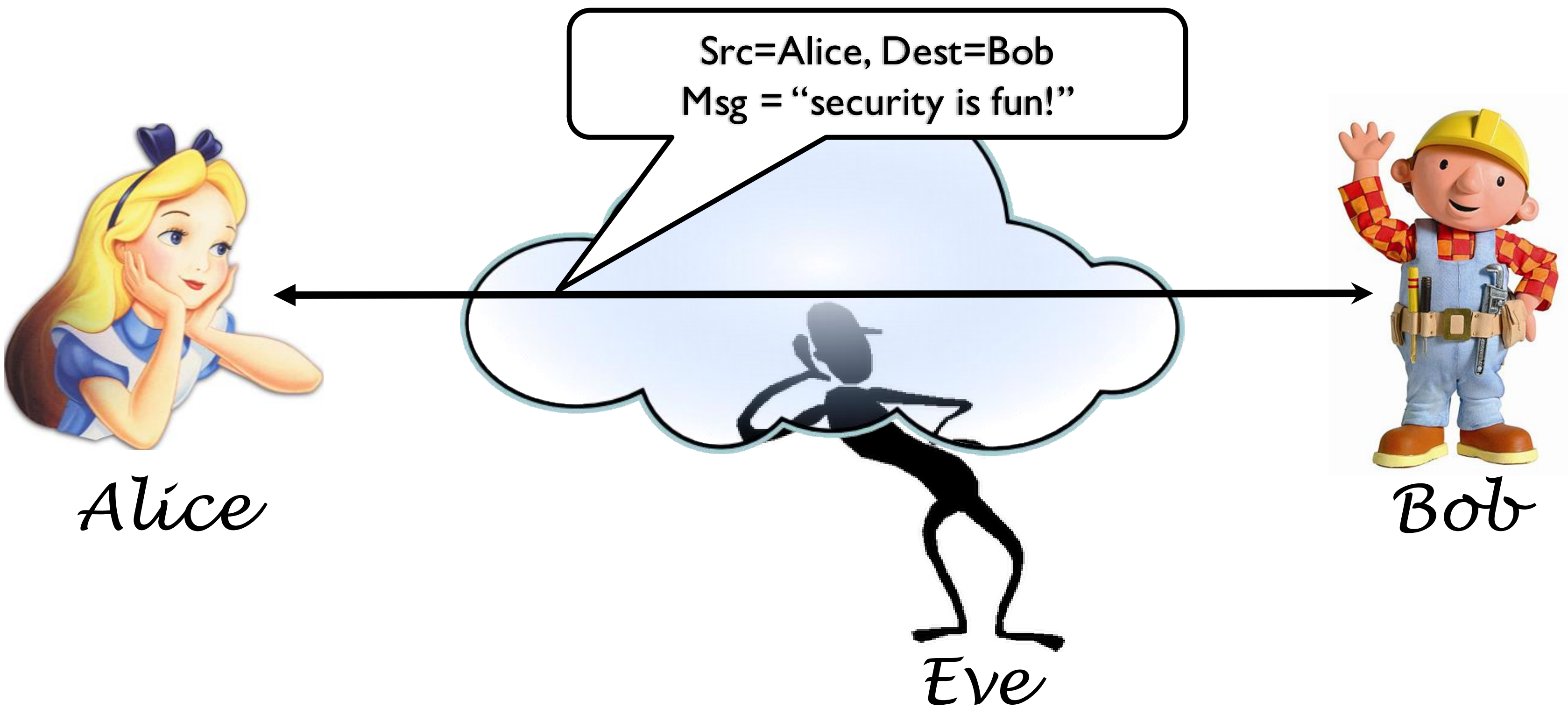
Class Notes and Clarifications

- Using Latex
  - Learn it!
  - Useful forever!
- Research proposal due today!
  - Any questions?

What encryption does and does not

- Does:
  - confidentiality
- Doesn’t do:
  - data integrity
  - source authentication
- **Need:** ensure that data is not altered and is from an authenticated source

Principals



Man-in-the-Middle (MitM) attack

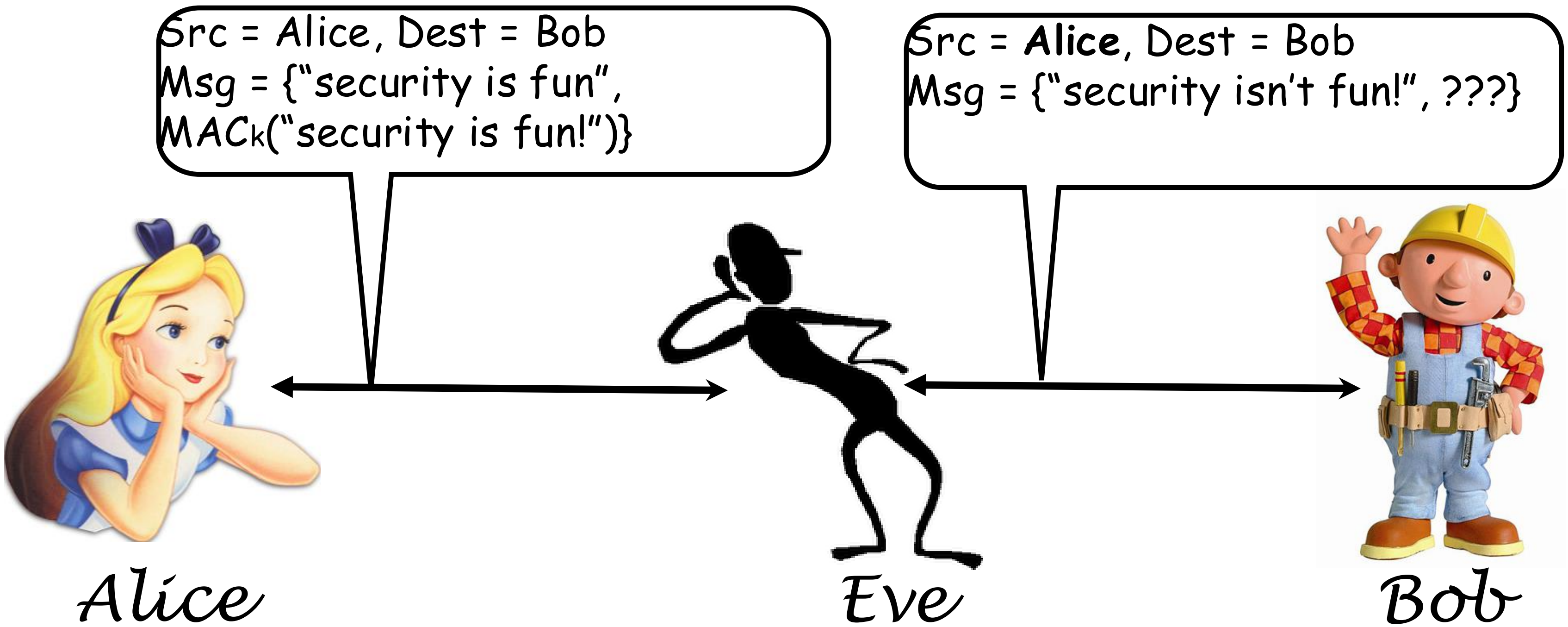
- For confidentiality, just encrypt.
- How do we ensure integrity?

Message Authentication Codes (MACs)

- MACs provide message **integrity** and **authenticity**
- $MAC_K(M)$ – use symmetric encryption to produce **short sequence of bits** that depends on both the message (M) and the key (K)
- MACs should be resistant to **existential forgery**: Eve should not be able to produce a valid MAC for a message M' without knowing K
- To provide confidentiality, authenticity, and integrity of a message, Alice sends
  - **$E_K(M, MAC_K(M))$** where $E_K(X)$ is the encryption of X using key K
- Proves that M was encrypted (confidentiality and integrity) by someone who knew K (authenticity)

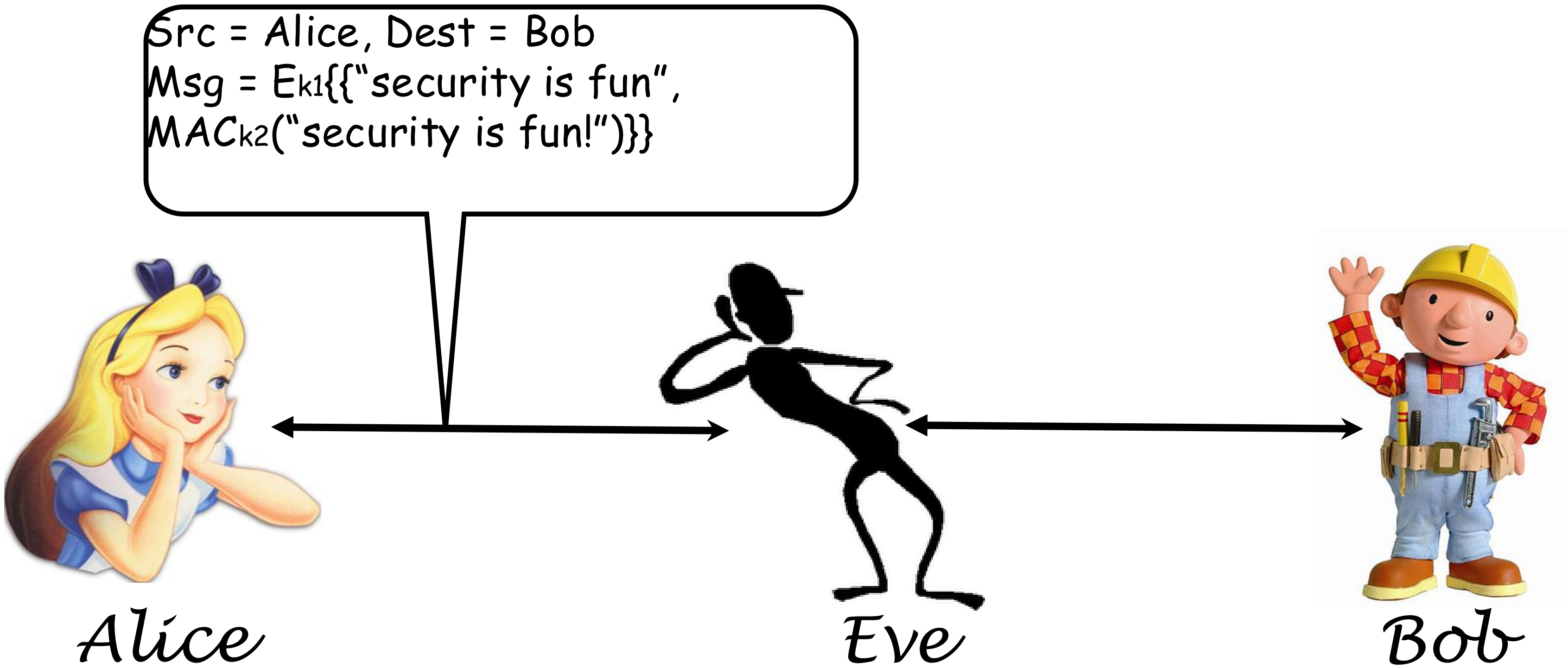
Why are we sending M?

Message Authenticity



Without knowledge of *k*, Eve can’t compute a valid MAC for her forged message!

Encryption and Message Authenticity



Without knowing *k1*,
Eve can't read Alice's message.

Without knowing *k2*, Eve can't compute a valid MAC for her forged message!

Cryptographic Hash Functions

- **Hash function** `h`: deterministic one-way function that takes as input an arbitrary message `M` (sometimes called a *preimage*) and returns as output `h(M)`, a small fixed length *hash* (sometimes called a *digest*)
- Hash functions should have the following two properties:
  - *compression*: reduces arbitrary length string to fixed length hash
  - *ease of computation*: given message `M`, `h(M)` is easy to compute

Hash functions are usually fairly inexpensive

(i.e., compared with public key cryptography)

```
~ openssl speed sha256
Doing sha256 ops for 3s on 16 size blocks: 33515432 sha256 ops in 3.00s
Doing sha256 ops for 3s on 64 size blocks: 29299431 sha256 ops in 2.99s
Doing sha256 ops for 3s on 256 size blocks: 19059503 sha256 ops in 3.00s
Doing sha256 ops for 3s on 1024 size blocks: 7433662 sha256 ops in 3.00s
Doing sha256 ops for 3s on 8192 size blocks: 1104810 sha256 ops in 3.00s
Doing sha256 ops for 3s on 16384 size blocks: 559814 sha256 ops in 2.99s
version: 3.3.1
built on: Tue Jun  4 12:53:04 2024 UTC
options: bn(64,64)
compiler: clang -fPIC -arch arm64 -O3 -Wall -DL_ENDIAN -DOPENSSL_PIC -D_REENTRANT
CPUINFO: OPENSSL_armcap=0x987d
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes  163
sha256          178748.97k   627145.01k  1626410.92k  2537356.63k  3016867.84k  30

~
```

Why might hashes be useful?

- **Message authentication codes (MACs):**
  - e.g.: `MAC_K(M) = h(K|M)`
    (but don't do this, use HMAC instead)
- **Modification detection codes:**
  - detect modification of data
  - any change in data will cause change in hash

Prof. Pedantic proposes the following hash function, arguing that it offers both compression and ease of computation.

- h(M) = 0 if the number of 0s in M is divisible by 3
- h(M) = 1 otherwise

Why is this a lousy crypto hash function?

Cryptographic Hash Functions

- Properties of good <u>cryptographic</u> hash functions:
  - **preimage resistance:** given digest y, computationally infeasible to find preimage x' such that h(x')=y
    (also called "one-way property")
  - **2nd-preimage resistance:** given preimage x, computationally infeasible to find preimage x' such that h(x)=h(x')
    (also called "weak collision resistance")
  - **collision resistance:** computationally infeasible to find preimages i,j such that h(i)=h(j)
    (also called "strong collision resistance")

Birthday Attack

- **Birthday Paradox:** chances that 2+ people share birthday in group of 23 is > 50%.
- General formulation
  - function f() whose output is uniformly distributed over H possible outputs
  - Number of experiments Q(H) until we find a collision is approximately:

$$Q(H) \approx \sqrt{\frac{\pi}{2} H}$$

  - E.g.,

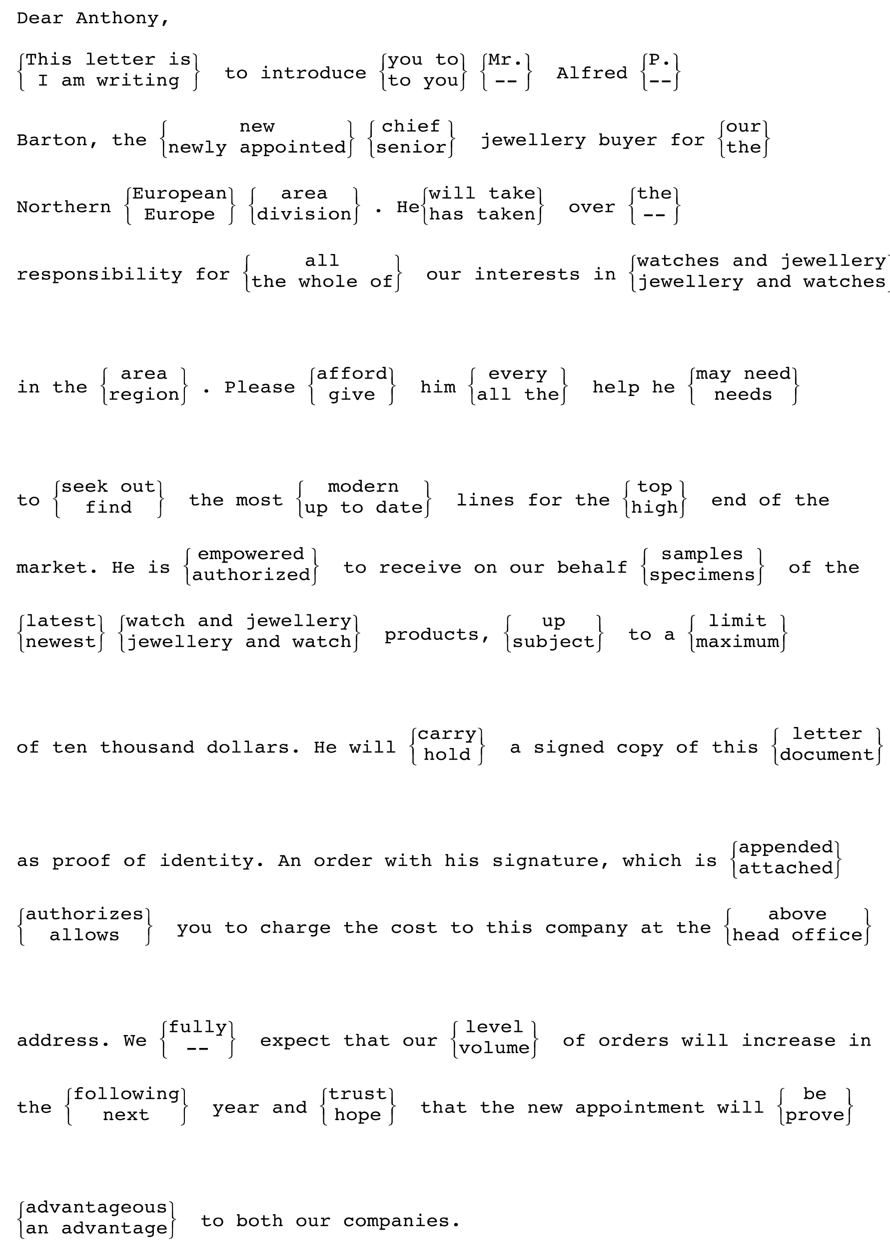
$$Q(365) \approx \sqrt{\frac{\pi}{2} 365} = 23.94$$

- Why is this relevant to hash sizes?

See: https://betterexplained.com/articles/understanding-the-birthday-paradox/

Practical Implications

- Choosing two messages that have the same hash h(x) = h(x') is more practical than you might think.
- Example attack: secretary is asked to write a "bad" letter, but wants to replace with a "good" letter.
  - Boss signs the letter after reading
- Find collision between 2^37 'good' vs 2^37 'bad' letters


Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --} Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the} Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --} responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches} in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs} to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the {latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum} of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document} as proof of identity. An order with his signature, which is {appended / attached} {authorizes / allows} you to charge the cost to this company at the {above / head office} address. We {fully / --} expect that our {level / volume} of orders will increase in the {following / next} year and {trust / hope} that the new appointment will {be / prove} {advantageous / an advantage} to both our companies.

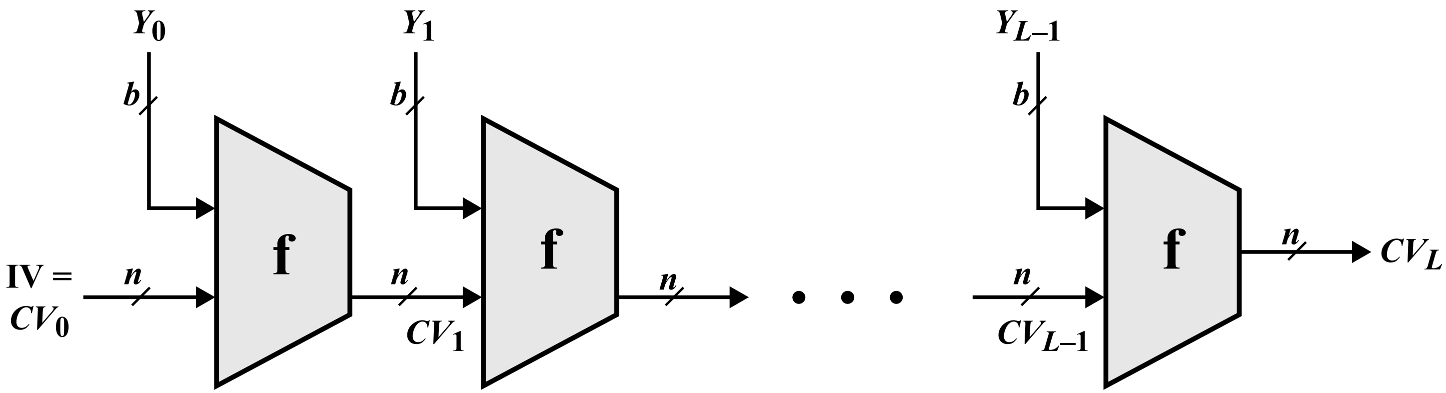
Figure 11.7 A Letter in 2^{37} Variations

(from Stallings, Crypto and Net Security)

Some common cryptographic hash functions

- MD5 (128-bit digest) [don’t use this]
- SHA-1 (160-bit digest) [stop using this*]
- SHA-256 (256-bit digest)
- SHA-512 (512-bit digest)
- SHA-3 [recent competition winner]

General Structure of Hash



IV	=	Initial value	L	=	number of input blocks
CV_i	=	chaining variable	n	=	length of hash code
Y_i	=	ith input block	b	=	length of input block
f	=	compression algorithm			

(from Stallings, Crypto and Net Security)

Message Extension Attack

- Why is $MAC_k(M) = H(k|M)$ bad?
- How can Eve append M’ to M?
  - Goal: compute H(k|M|M’) without knowing k
- Solution: Use H(k|M) as IV for next f iteration in H()

A Better MAC

- Objectives
  - Use available hash functions without modification
  - Easily replace embedded hash function as more secure ones are found
  - Preserve original performance of hash function
  - Easy to use

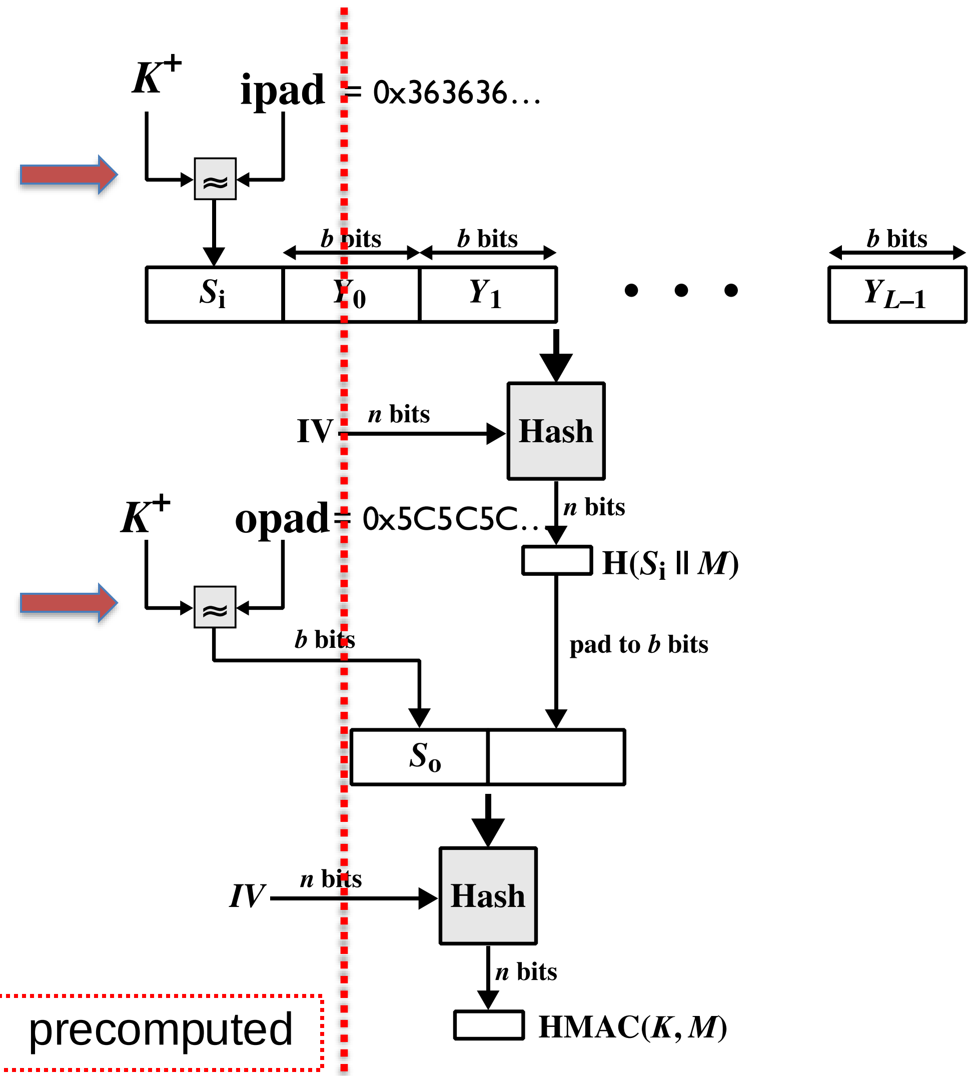
HMAC

HMAC(k, M)

H(k⊕opad || H(k⊕ipad || M))

hash2 ***hash1***

- Attacker cannot extend MAC as before
  - Try it out!



(from Stallings, Crypto and Net Security)

Basic truths of cryptography

- Cryptography is not frequently the source of security problems
  - Algorithms are well known and widely studied
  - Vetted through crypto community
  - Avoid any “proprietary” encryption
  - Claims of “new technology” or “perfect security” are almost assuredly **snake oil**

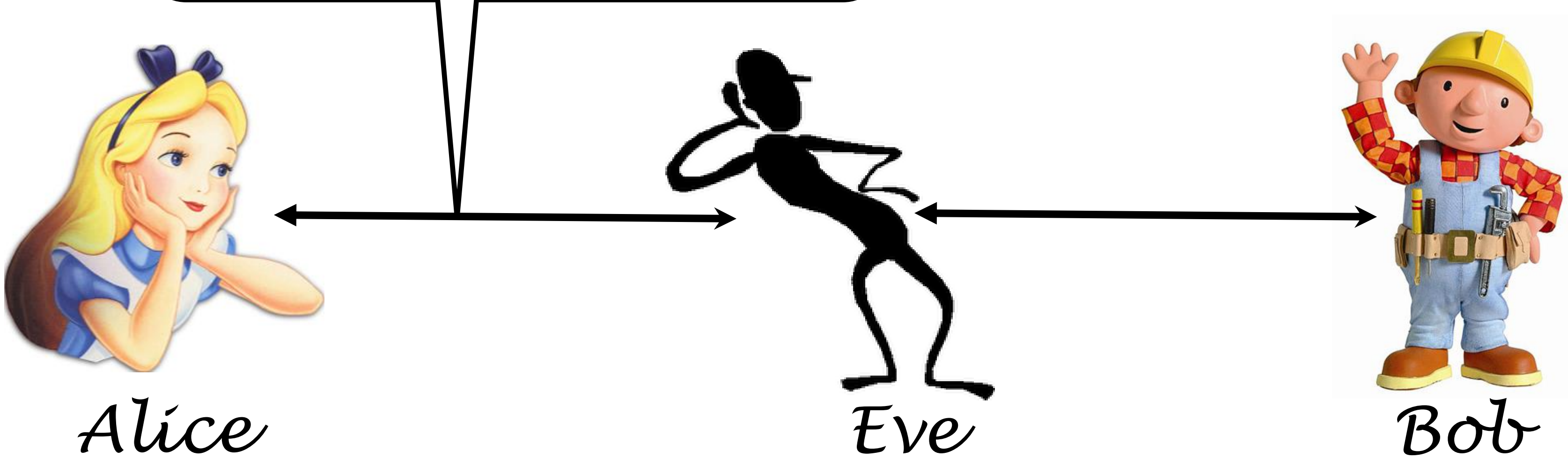
Building systems/apps with cryptography

- Use quality libraries
  - SSLeay, cryptolib, openssl
  - Find out what cryptographers think of a package before using it
- Code review like crazy
- Educate yourself on how to use library
  - Understand caveats by original designer and programmer

Encryption and Message Authenticity

What's the hard part?

Src = Alice, Dest = Bob
Msg = E_{k1}{{"network security is fun",
MAC_{k2}("network security is fun!")}}



Without knowing *k1*, Eve can't read Alice's message.

Without knowing *k2*, Eve can't compute a valid MAC for her forged message.

Private-key crypto is like a door lock

Why?

Public Key Crypto
(10,000 ft view)

- Separate keys for encryption and decryption
  - Public key: anyone can know this
  - Private key: kept confidential
- Anyone can encrypt a message to you using your public key
- The private key (kept confidential) is required to decrypt the communication
- Alice and Bob no longer have to have *a priori* shared a secret key

Public Key Cryptography

- Each key pair consists of a public and private component: k^+ (public key), k^- (private key)

$$D_{k^-}(E_{k^+}(m)) = m$$

- Public keys are distributed (typically) through public key certificates
  - Anyone can communicate secretly with you ***if they have your certificate***

RSA
(Rivest, Shamir, Adelman)

- The dominant public key algorithm
  - The algorithm itself is conceptually simple
  - Why it is secure is very deep (number theory)
  - Uses properties of exponentiation modulo a product of large primes

"A method for obtaining Digital Signatures and Public Key Cryptosystems", Communications of the ACM, Feb. 1978.

Modular Arithmetic

- Integers $Z_n = \{0, 1, 2, ..., n-1\}$
- x mod n = remainder of x divided by n
  - 5 mod 13 = 5
  - 13 mod 5 = 3
- y is **modular inverse** of x iff **xy mod n = 1**
  - E.g. Z_{11} -> 4 is inverse of 3, 5 is inverse of 9, 7 is inverse of 8
- If **n is prime**, then Z_n has modular inverses for all integers except 0

Euler's Totient Function

- **coprime**: having no common positive factors other than 1 (also called **relatively prime**)
  - 16 and 25 are coprime
  - 6 and 27 are not coprime
- **Euler's Totient Function**: Φ(n) = number of integers less than or equal to n that are coprime with n

$$\Phi(n) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$$

where product ranges over distinct primes dividing n

- If m and n are coprime, then Φ(mn) = Φ(m)Φ(n)

- If m is prime, then Φ(m) = m - 1

Euler's Totient Function

$$\Phi(n) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$$

$$\Phi(18) = \Phi(3^2 \cdot 2^1) = 18(1 - \frac{1}{3})(1 - \frac{1}{2}) = 6$$

For primes and co-primes:

If m and n are coprime, then $\Phi(mn) = \Phi(m)\Phi(n)$

If m is prime, then $\Phi(m) = m - 1$

RSA Key Generation

Example:

1. Choose distinct primes p and q
2. Compute $n = pq$
3. Compute $\Phi(n) = \Phi(pq) = \Phi(p)\Phi(q) = (p-1)(q-1)$
4. Randomly choose $1 < e < \Phi(pq)$ such that e and $\Phi(pq)$ are coprime. e is the **public key exponent**
5. Compute $d = e^{-1} \bmod (\Phi(pq))$. d is the **private key exponent**

let $p=3$, $q=11$

$n=33$

$\Phi(pq)=(3-1)(11-1)=20$

let $e=7$

$ed \bmod \Phi(pq) = 1$

$7d \bmod 20 = 1$

$d = 3$

RSA Encryption/Decryption

- Public key k^+ is {e,n} and private key k^- is {d,n}
- Encryption and Decryption

$$E_{k+}(M) : \text{ciphertext} = \text{plaintext}^{e} \bmod n$$

$$D_{k-}(\text{ciphertext}) : \text{plaintext} = \text{ciphertext}^{d} \bmod n$$

- Example
  - **Public key (7,33), Private Key (3,33)**
  - Plaintext: 4

  - E({7,33},4) = 4^7 mod 33 = 16384 mod 33 = 16
  - D({3,33},16) = 16^3 mod 33 = 4096 mod 33 = 4

Is RSA Secure?

- $\{e,n\}$ is public information
- If you could factor n into $p*q$, then
  - could compute $\phi(n) = (p-1)(q-1)$
  - could compute $\underline{d = e^{-1} \bmod \phi(n)}$
  - would know the private key $<d,n>$!
- But: factoring large integers is hard!
  - classical problem worked on for centuries; no known reliable, fast method

Security (Cont’d)

- At present, key sizes of 1024 bits are considered to be secure, but 2048 bits is better
- Tips for making *n* difficult to factor
  1. *p* and *q* lengths should be similar (ex.: ~500 bits each if key is 1024 bits)
  2. both (p-1) and (q-1) should contain a “large” prime factor
  3. gcd(p-1, q-1) should be “small”
  4. *d* should be larger than $n^{1/4}$

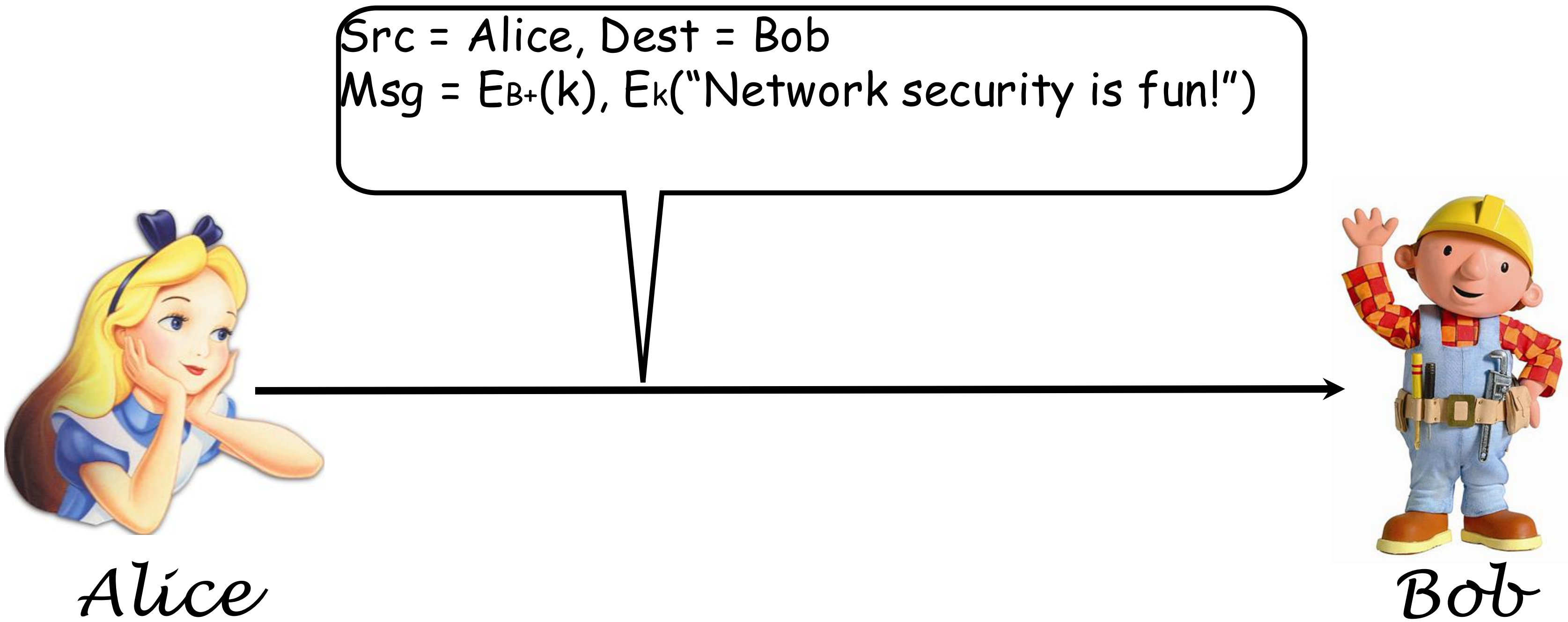
RSA

- Most public key systems use at least 1,024-bit keys
  - Key size not comparable to symmetric key algorithms
- RSA is *much slower* than most symmetric crypto algorithms
  - AES: ~161 MB/s
  - RSA: ~82 KB/s
  - This is **too** slow to use for modern network communication!
  - Solution: Use **hybrid model**

Hybrid Cryptosystems

- In practice, public-key cryptography is used to secure and distribute *session keys*.
- These keys are used with symmetric algorithms for communication.
- Sender generates a random session key, encrypts it using receiver's public key and sends it.
- Receiver decrypts the message to recover the session key.
- Both encrypt/decrypt their communications using the same key.
- Key is destroyed in the end.

Hybrid Cryptosystems



(B^+,B^-) is Bob’s long-term public-private key pair.

k is the session key; sometimes called the **ephemeral key**.

Public Key Cryptography

- Each key pair consists of a public and private component: k^+ (public key), k^- (private key)

$$D_{k^-}(E_{k^+}(m)) = m$$

What happens if we flip the order?

Encryption using private key

- Encryption and Decryption

  $E_{k-}(M)$: ciphertext = plaintext^{d} mod n

  D_{k+}(ciphertext) : plaintext = ciphertext^{e} mod n

- E.g.,
  - E({3,33},4) = 4^3 mod 33 = 64 mod 33 = 31
  - D({7,33},31) = 31^7 mod 33 = 27,512,614,111 mod 33 = 4
- Q: *Why encrypt with private key?*
  - *Non Repudiation!*

Digital Signatures

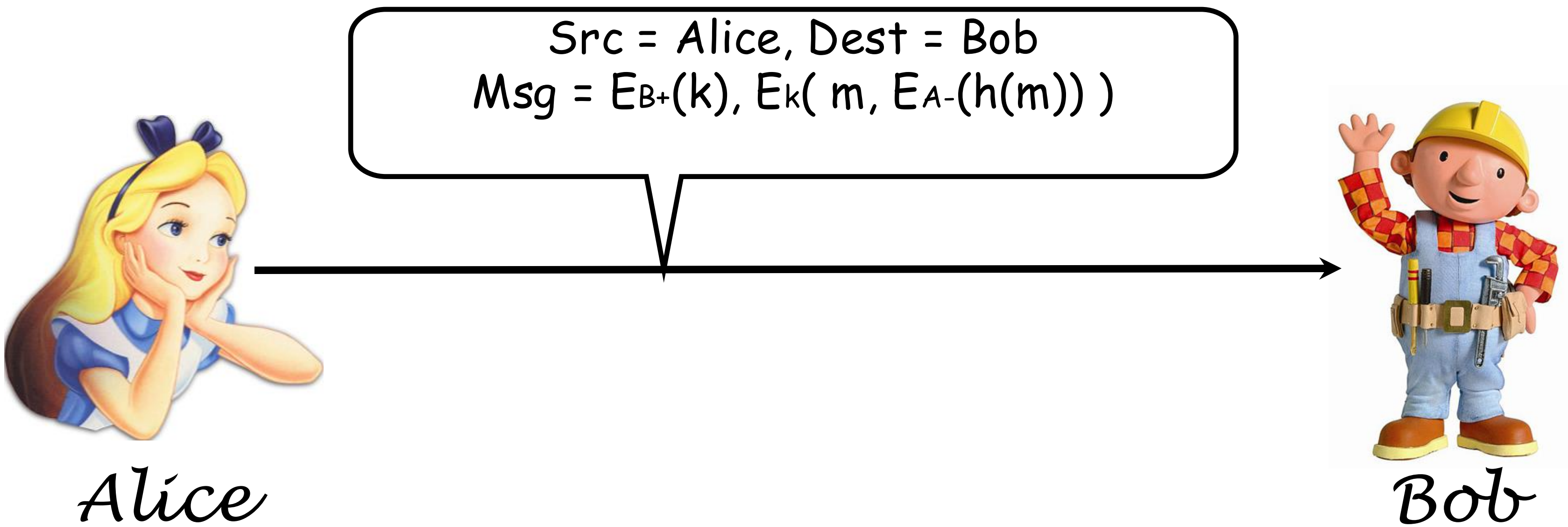
- A digital signature serves the same purpose as a real signature.
  - It is a mark that only sender can make
  - Other people can easily recognize it as belonging to the sender
- Digital signatures must be:
  - **Unforgeable**: If Alice signs message M with signature S, it is impossible for someone else to produce the pair (M, S).
  - **Authentic**: If Bob receives the pair (M, S) and knows Alice's public key, he can check ("verify") that the signature is really from Alice
  - Example: Code signing

How can Alice *sign* a digital document?

- Digital document: M
- Since RSA is slow, hash M to compute digest: m = h(M)
- Signature: Sig(M) = $E_{k-}(m) = m^d \bmod n$
  - Since only Alice knows k^-, only she can create the signature
- To verify: Verify(M,Sig(M))
  - Bob computes h(M) and compares it with D_{k+}(Sig(M))
  - Bob can compute D_{k+}(Sig(M)) since he knows k^+ (Alice's public key)
  - If and only if they match, the signature is verified (otherwise, verification fails)

Putting it all together

Define m = "Network security is fun!"



(A^+, A^-) is Alice's long-term public-private key pair.

(B^+,B^-) is Bob's long-term public-private key pair.

k is the session key; sometimes called the **ephemeral key**.

Birthday Attack and Signatures

- Since signatures depend on hash functions, they also depend on the hash function's collision resistance
- Don't use MD5, and start moving away from SHA1

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --} Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the} Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --} responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches} in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs} to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the {latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum} of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document} as proof of identity. An order with his signature, which is {appended / attached} {authorizes / allows} you to charge the cost to this company at the {above / head office} address. We {fully / --} expect that our {level / volume} of orders will increase in the {following / next} year and {trust / hope} that the new appointment will {be / prove} {advantageous / an advantage} to both our companies.

Figure 11.7 A Letter in 2^{37} Variations

(from Stallings, Crypto and Net Security)

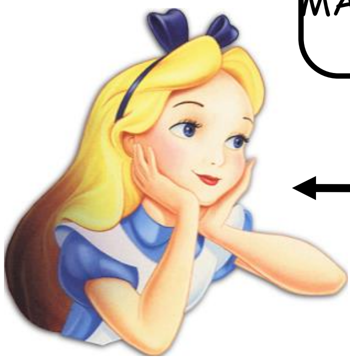
Properties of a Digital Signature

- **No forgery possible:** No one can forge a message that is purportedly from Alice
- **Authenticity check:** If you get a signed message you should be able to verify that it's really from Alice
- **No alteration/Integrity:** No party can undetectably alter a signed message
- Provides authentication, integrity, and **non-repudiation** (cannot deny having signed a signed message)

Non-Repudiation

Src = Alice, Dest = Bob
Msg = {"network security is fun", MAC_k("network security is fun!")}

MAC

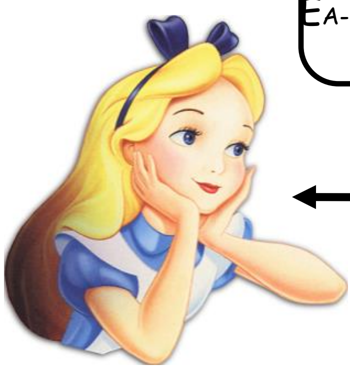


Alice

Bob

Which of these offer non-repudiation?

Src = Alice, Dest = Bob
Msg = {"network security is fun", E_{A-}(h("network security is fun!"))}



Alice

Bob