CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Franzi Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

- My office hours
  - 11/6 (Wed), 1:30pm, CSE1 678 (small room, unfortunately)
  - 11/13 (Wed), 11:30am, CSE1 403
  - 11/20 (Wed), 2:30pm, CSE1 403
  - 11/27 (Wed), None
  - 12/4 (Wed), 12:30pm, CSE1 403
- TA office hours today as normal, but different TAs
- HW 2 available
- Quiz section next week: Lab 2

Which Property Do We Need?

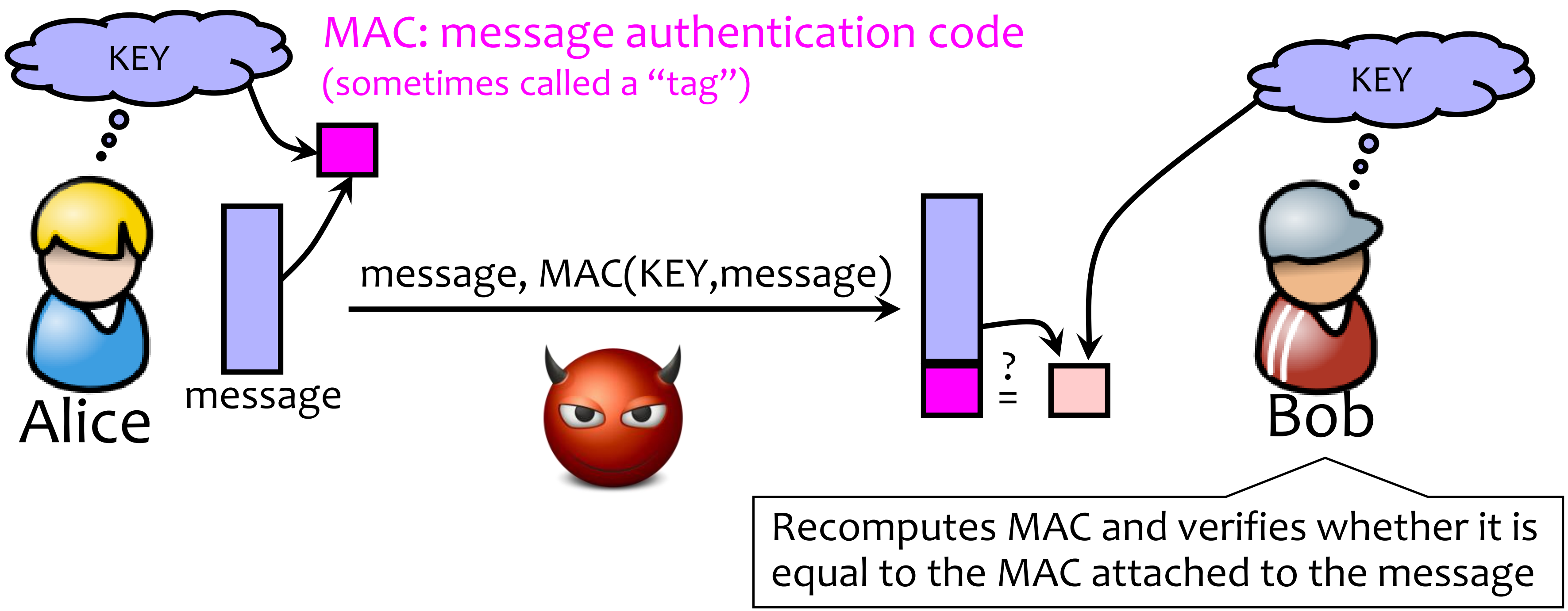
- UNIX passwords stored as hash(password)
  - **One-wayness:** hard to recover the/a valid password
- Financial transactions
  - **Weak collision resistance** (first example)
  - **Collision resistance** (second example)
- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - **One-wayness:** rival bidders should not recover B (this may mean that she needs to hash some randomness with B too)
  - **Collision resistance:** Alice should not be able to change her mind to bid B' such that H(B)=H(B')

Common Hash Functions

- MD5 – Don't use!
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm) – Don't use!
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Theoretically broken 2005; practical attack 2017!
- SHA-256, SHA-512, SHA-224, SHA-384
- SHA-3: standard released by NIST in August 2015

Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.



Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

HMAC

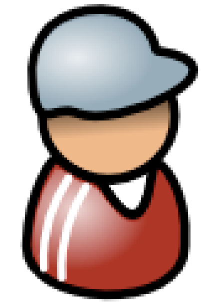
- Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for Ipsec
- Construction
  - HMAC(K,M) = Hash(K xor OPAD, Hash(K xor IPAD, M))
- Why not block ciphers? (At the time it was designed)
  - Hashing is faster than block ciphers in software
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

Challenge Question

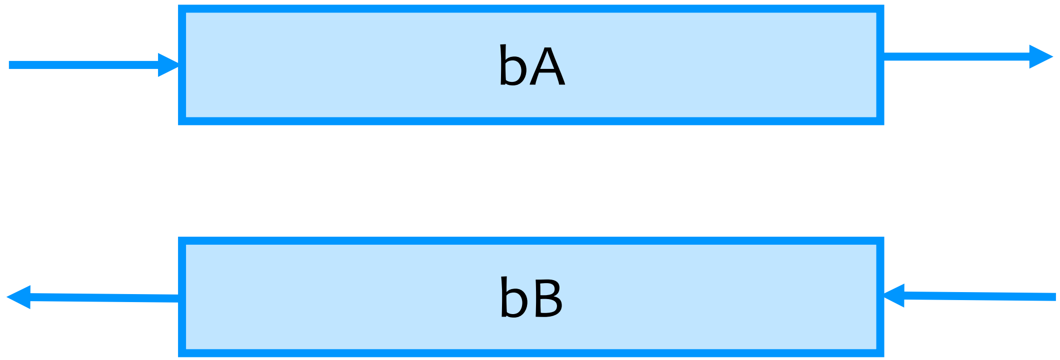
- Alice and Bob are both cryptographers, and they are talking on the phone. They want to randomly flip a coin. If they were together, in person, they would flip a real coin and see if it was Heads or Tails. But they are not together, in person, and they don't trust each other enough to have one of them flip a coin and tell the other person the answer.

- Using the techniques we've discussed so far in class, how can Alice and Bob effectively flip a random coin together, over the phone, such that they both trust the answer even though they don't trust each other?

Not a Solution



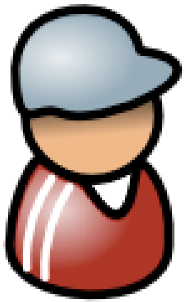
Pick bit bA at random



Pick bit bB at random

Both compute random bit as bA xor bB

Why not a solution? Because Bob can pick bB such that bA xor bB is whatever outcome Bob wants



Pick bit bA at random

C1 = Locked box with bA

Pick bit bB at random

C2 = Locked box with bB

Info to open first box C1

Now knows bA

Info to open second box C2

Now knows bB

Both compute random bit as bA xor bB

Challenge Question

- Alice and Bob are both cryptographers, and they are talking on the phone. They want to randomly flip a coin. If they were together, in person, they would flip a real coin and see if it was Heads or Tails. But they are not together, in person, and they don't trust each other enough to have one of them flip a coin and tell the other person the answer.

- Using the techniques we've discussed so far in class, how can Alice and Bob effectively flip a random coin together, over the phone, such that they both trust the answer even though they don't trust each other?

Pick bit bA at random

|| denotes concatenation

Pick RA as long random string

C1 = H(bA || RA)

Pick bit bB at random

C2 = H(bB || RB)

Pick RB as long random string

Send bA || RA

Verify that has of message equals C1

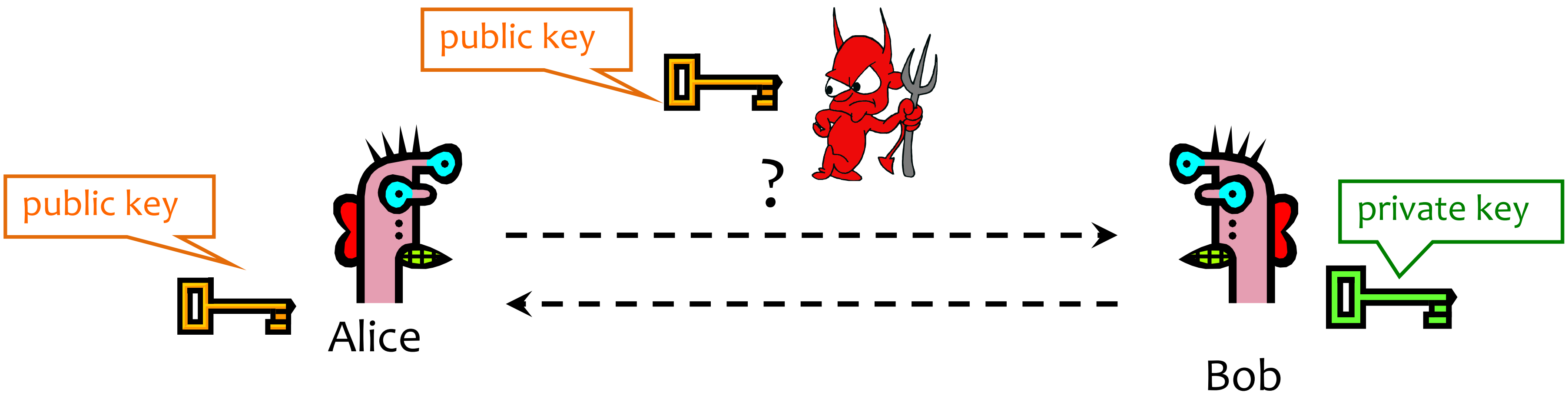
Send bB || RB

Verify that has of message equals C2

Both compute random bit as bA xor bB

Back to RSA

Public Key Crypto: Basic Problem



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

Goals: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 1024 bits each (need primality testing, too)
  - Compute **n**=pq and φ(**n**)=(p-1)(q-1)
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, e=3 or $e=2^{16}+1=65537$
  - Compute unique d such that $ed \equiv 1 \bmod \varphi(n)$
    - Modular inverse: $d \equiv e^{-1} \bmod \varphi(n)$

How to compute?

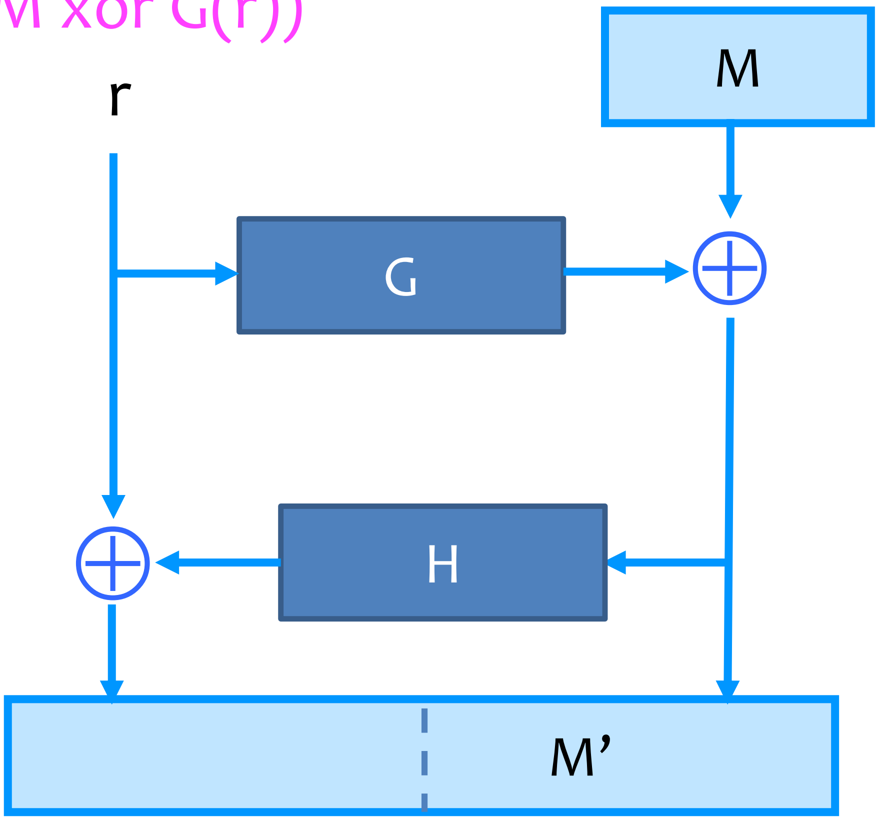
  - Public key = (e,n); private key = (d,n)
- Encryption of m (m a number between 0 and n-1): $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e \bmod n)^d \bmod n = m$

RSA + OAEP

- Plain RSA encryption malleable, e.g.,
  - Adversary sees $C1 = M1^e \bmod N$
  - Adversary sees $C2 = 2^e \bmod N$ // or any value Adversary wants
  - Adversary compute $C3 = C1 * C2 \bmod N$
  - Adversary sends C3 to Bob
  - Bob decrypts C3. Result is $C3^d \bmod N = (C1*C2)^d \bmod N = C1^d *C2^d \bmod N = 2*M1 \bmod N$
  - This structural property is undesirable / unexpected for a "secure" encryption scheme
- Also problems if M < cube root of N (if e=3)
- In practice, OAEP is used: instead of encrypting M, encrypt M xor G(r) ; r xor H(M xor G(r))
  - r is random and new each time, G and H are hash functions

OAEP as a Figure

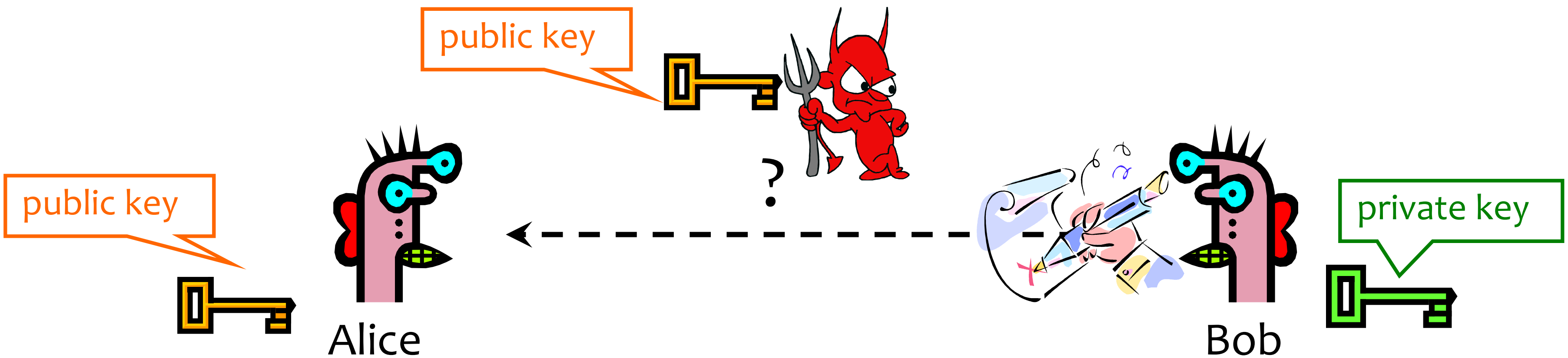
- M xor G(r) ; r xor H(M xor G(r))
- G, H hash functions



- C = (M')^e mod n
- Do you see how to decrypt C to recover M? (Side note, similar to DES internals)

Digital Signatures

Digital Signatures: Basic Idea



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

Goal: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed

RSA Signatures

- Public key is **(n,e)**, private key is **(n,d)**
- To sign message m: $s = m^d \bmod n$
  - Signing & decryption are same **underlying** operation in RSA
  - It's infeasible to compute **s** on **m** if you don't know **d**
- To verify signature s on message m:

  verify that $s^e \bmod n = (m^d)^e \bmod n = m$
  - "Just like encryption" (for RSA primitive)
  - Anyone who knows **n** and **e** (public key) can verify signatures produced with d (private key)
- "Just like encryption" in quotes!

RSA Signature Malleability

- Plain RSA signatures malleable, e.g.,
  - Adversary sees M1, $S1 = M1^d \bmod N$
  - Adversary sees M2, $S2 = M2^d \bmod N$
  - Adversary compute $S3 = S1 * S2 \bmod N$; $M3=M1*M2 \bmod N$
  - Adversary sends M3, S3 to Alice
  - Alice verifies signature of M3, S3. Via $S3^e \bmod N = (S1*S2)^e \bmod N = S1^e *S2^e \bmod N = M1*M2 \bmod N = M3$; signature verifies
  - Conclusion: Adversary can forge signature of M3 if sees signature for M1,M2
- In practice, also need padding & hashing
- Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

- Digital Signature Standard (DSS)
  - U.S. government standard (1991, most recent rev. 2013)
- Public key: $(p, q, g, y=g^x \bmod p)$, private key: x
- Signing is randomized
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from $g^x \bmod p$ (public key)

- **Important Note: Significant advantages in using elliptic curve groups – groups with some similar mathematical properties (i.e., are "groups") but have better security and performance (size) properties**

Stepping Back

Cryptography Summary Thus Far

- Goal: Privacy
  - Symmetric keys:
    - One-time pad, Stream ciphers
    - Block ciphers (e.g., DES, AES) → modes: EBC, CBC, CTR
  - Public key crypto (e.g., RSA)
- Goal: Integrity
  - MACs, often using hash functions (e.g, MD5, SHA-256)
- Goal: Privacy and Integrity
  - Encrypt-then-MAC
- Goal: Authenticity (and Integrity)
  - Digital signatures (e.g., RSA, DSS)