CSE 484: Computer Security and Privacy

Cryptography
[Asymmetric Cryptography]

Winter 2021

David Kohlbrenner

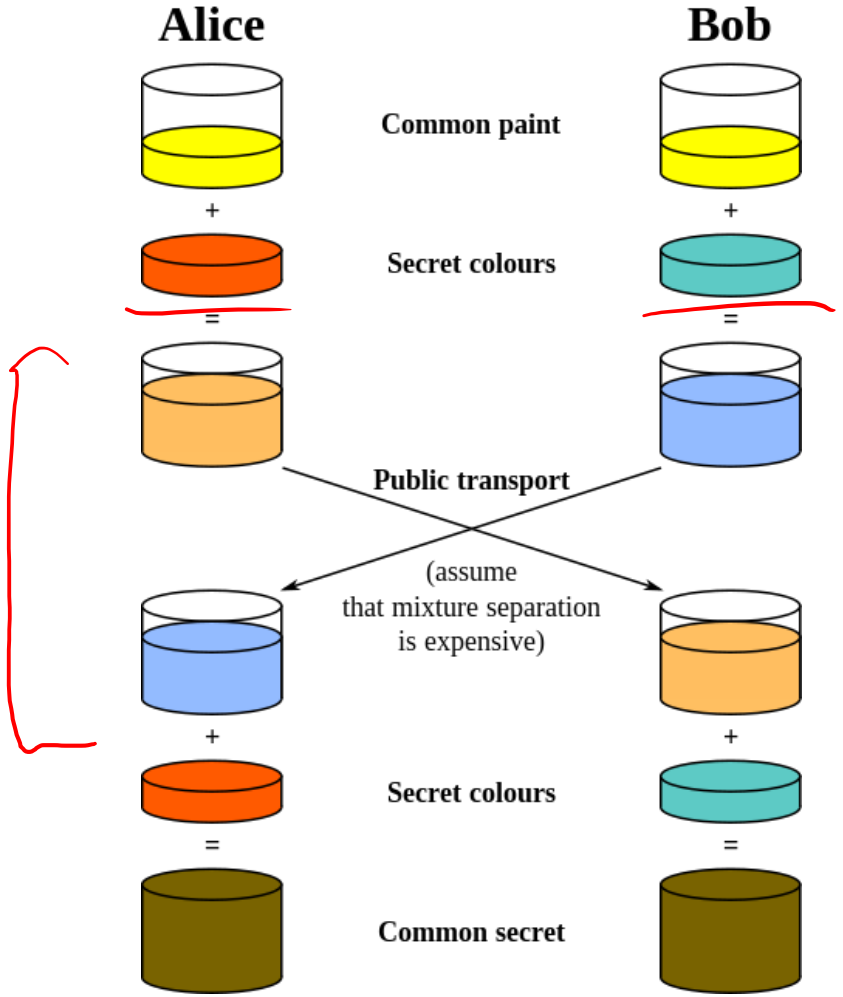
dkohlbre@cs.washington.edu

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

Admin

- **No class next Monday** (8^{th})
- **Lab 1 due tonight**
- In-class course feedback/evaluation on Friday

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

Send over public transport:

$g^x \bmod p$

$g^y \bmod p$

Common secret: $g^{xy} \bmod p$

[from Wikipedia]

Diffie-Hellman Caveats

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Common recommendation:
    - Choose p=2q+1, where q is also a large prime
    - Choose g that generates a subgroup of order q in Z_p*
  - Eavesdropper can't tell the difference between the established key and a random value
  - In practice, often hash g^{xy} *mod p*, and use the hash as the key
  - Can use the new key for symmetric cryptography
- Diffie-Hellman protocol (by itself) does not provide authentication (against active attackers)
  - Person in the middle attack (also called "man in the middle attack")

Remember our troubles with randomness?

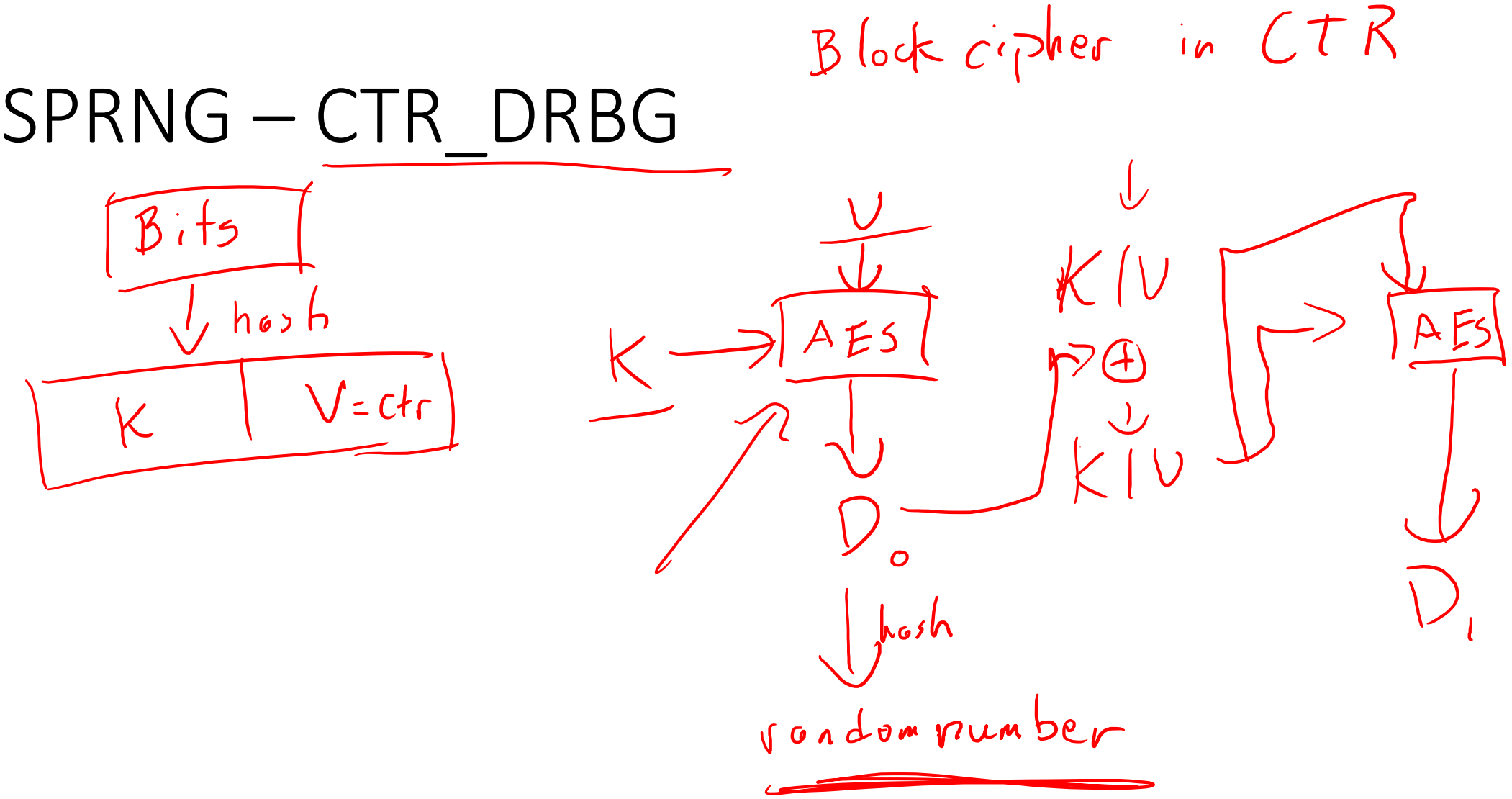
CSPRNGs in practice (or DRBG)

- Gather some good entropy (256 bits?)
- Use a block cipher/HMAC/Hash to 'stretch' this entropy
- Regularly mix in more entropy!

Check out NIST.SP.800-90

3 CSPRNGs

CSPRNG – CTR_DRBG



Why does this work for CSPRNGs?

- To 'break' the CSPRNG (that is, predict the next output)
  - Must know state of CSPRNG (key, inputs)
  - Requires breaking the security of the primitive!

- Your CSPRNG is just as secure as the scheme you use the output for!
  - Never 'loses entropy', same guarantee as block cipher

- Why mix in new entropy?
  - Can't hurt, prevents a single bug from breaking the future

CSPRNGs gone bad

4th rec. CSPRNG

2004 - 2013?

- DUAL_EC_DRBG – Dual Elliptic Curve Deterministic Random Bit Generator
  - CSPRNG based on elliptic curve math
  - NSA designed

juniper VPNs

- DUAL_EC_DRBG has a *backdoor*
  - Special mathematical construction that allows recovery of state!

- Remember DES's s-boxes?
  - This is the opposite

Stepping Back: Asymmetric Crypto

- We've just seen session key establishment
  - Can then use shared key for symmetric crypto → cause fast
- Next: public key encryption
  - For confidentiality
- Then: digital signatures
  - For authenticity

Requirements for Public Key Encryption

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
- Encryption: given plaintext M and public key PK, easy to compute ciphertext $C=E_{PK}(M)$
- Decryption: given ciphertext $C=E_{PK}(M)$ and private key SK, easy to compute plaintext M
  - Infeasible to learn anything about M from C without SK
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Encrypt

Some Number Theory Facts

- Euler totient function φ(n) (n≥1) is the number of integers in the [1,n] interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
  - Easy to compute for primes: φ(p) = p-1
  - Note that φ(ab) = φ(a) φ(b) if a & b are relatively prime

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 φ(n)
    - Typically, **e=3** or **e=2^{16}+1=65537**
  - Compute unique **d** such that ed ≡ 1 mod φ(n)
    - Modular inverse: $d \equiv e^{-1} \bmod \varphi(n)$
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

How to compute?

Why is RSA Secure?

n = p q

eff key strength
1024 bits = 80 bits of sec.

- RSA problem: given c, n=pq, and e such that gcd(e, φ(n))=1, find m such that $m^e = c \bmod n$
  - In other words, recover m from ciphertext c and public key (n,e) by taking e^{th} root of c modulo n
  - There is no known efficient algorithm for doing this
- Factoring problem: given positive integer n, find primes $p_1, \ldots, p_k$ such that $n = p_1^{e1} p_2^{e2} \ldots p_k^{ek}$

- If factoring is easy, then RSA problem is easy (knowing factors means you can compute d = inverse of e mod (p-1)(q-1))
  - It may be possible to break RSA without factoring n -- but if it is, we don't know how

RSA Encryption Caveats

$M^e \mod N$

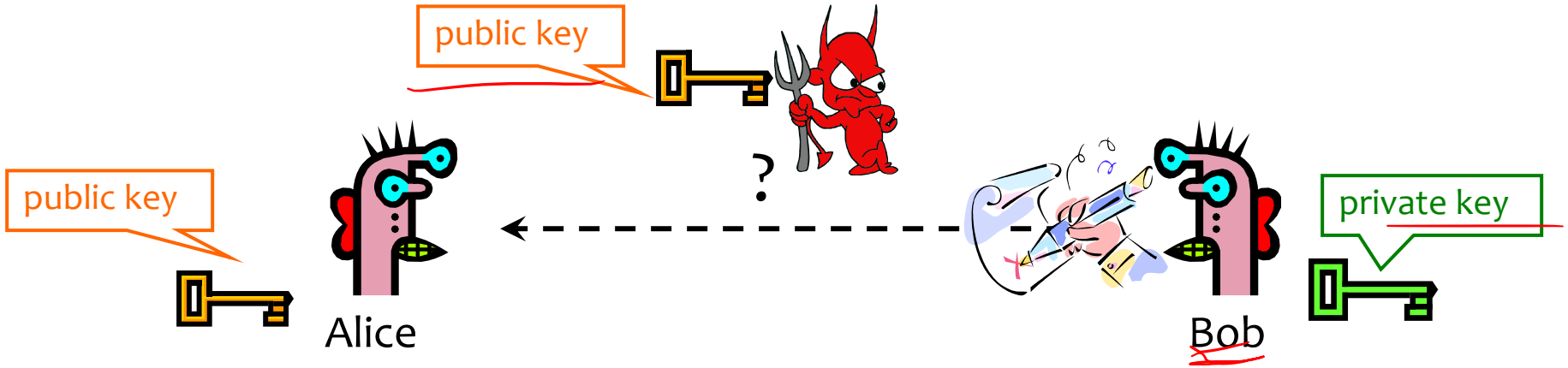
- Encrypted message needs to be interpreted as an integer less than n
- Don't use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
- Plain RSA also does not provide integrity
  - Can tamper with encrypted messages

In practice, OAEP is used: instead of encrypting M, encrypt

$M \oplus G(r) \,||\, r \oplus H(M \oplus G(r))$

- r is random and fresh, G and H are hash functions

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

e d

$M^{e^d} = M \bmod N$

$M^{d^e} \bmod N = M$

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)
- 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
- 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)
- Again: We've discussed discrete logs modulo integers; significant advantages to using elliptic curve groups instead.

Cryptography Summary

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

Want More Crypto?

- Some suggestions:
  - CSE 490C (Rachel Lin): https://courses.cs.washington.edu/courses/cse490c/20au/
  - Stanford Coursera (Dan Boneh): https://www.coursera.org/learn/crypto