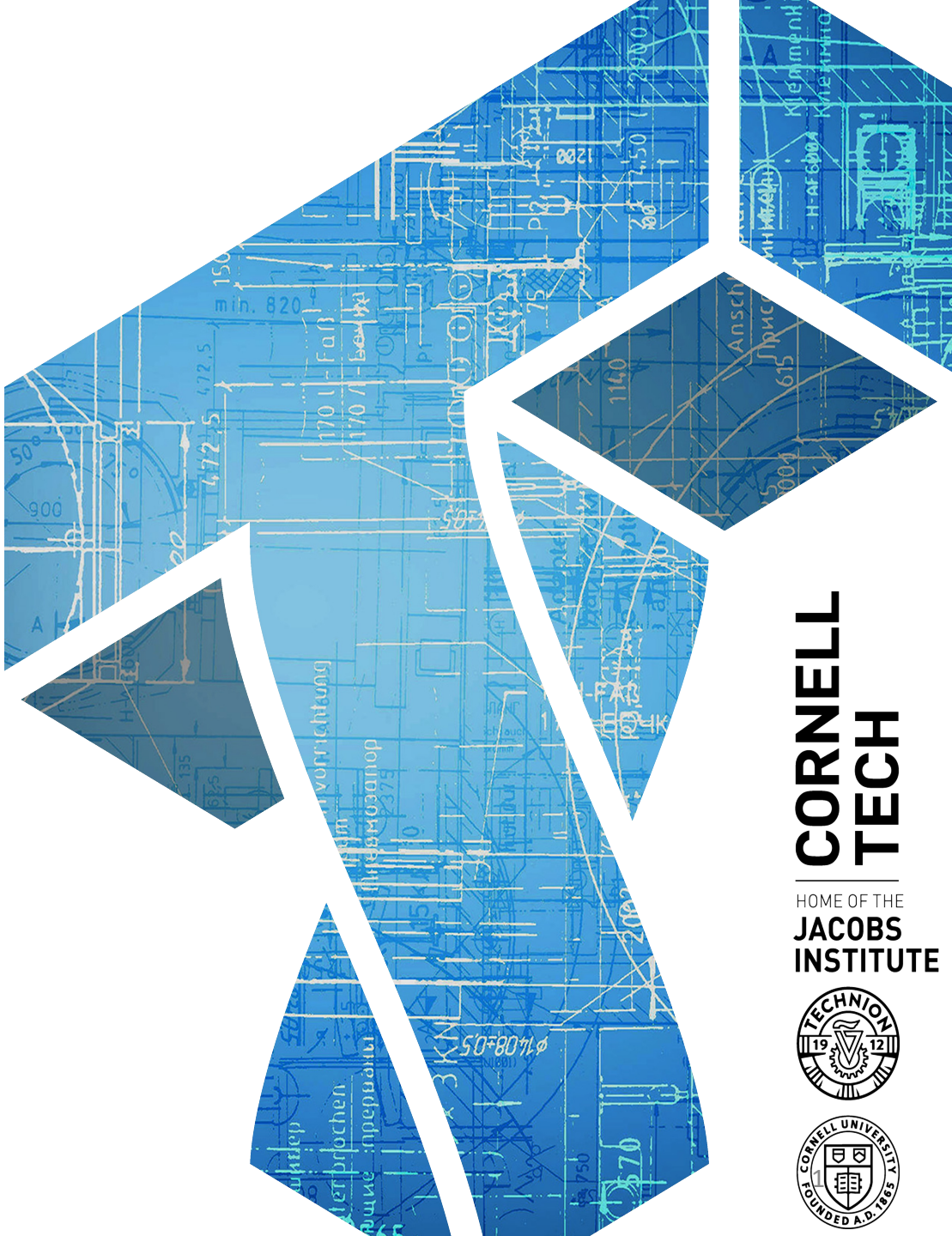
CS 5830
Cryptography

Instructor: Tom Ristenpart
TAs: Yan Ji, Sanketh Menda



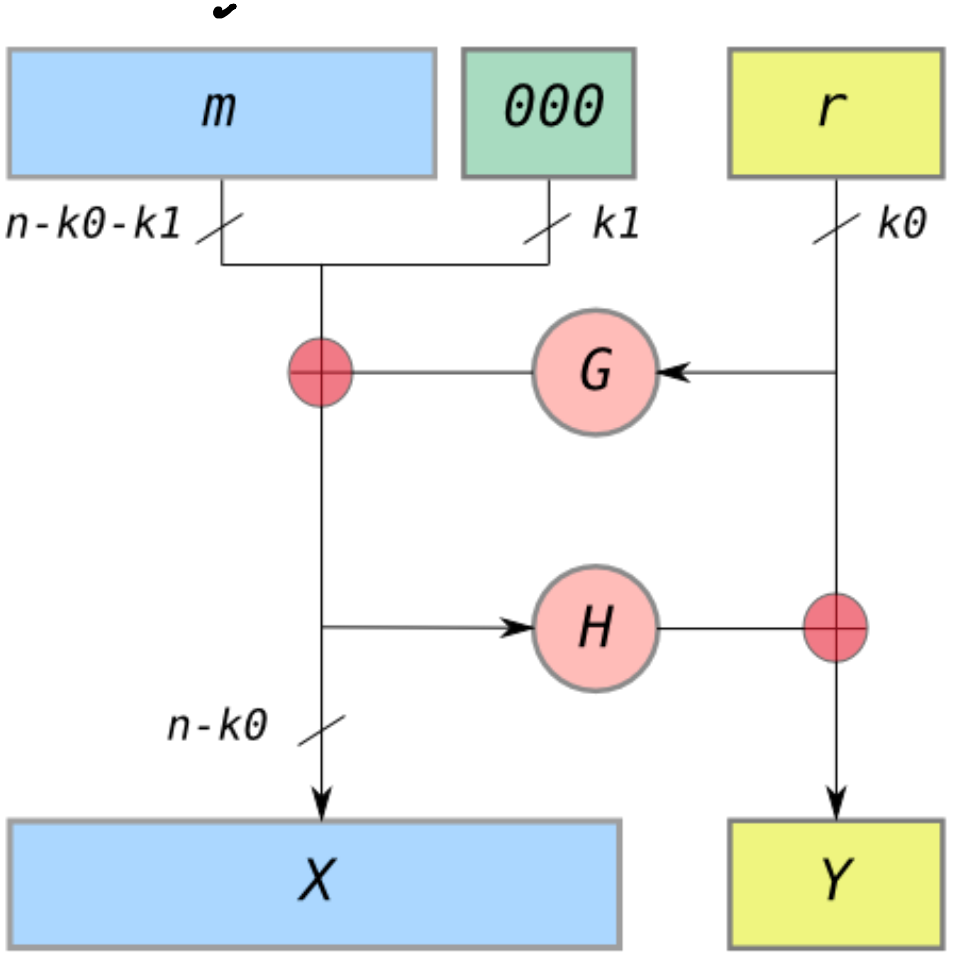
CORNELL TECH
HOME OF THE JACOBS INSTITUTE

CCA Attacks against PKE

- Ad-hoc fix to Bleichanbacher:
  - Don’t leak whether padding was wrong or not
  - This is harder than it looks (timing attacks, control-flow side channel attacks, etc.)
- Better:
  - use chosen-ciphertext secure encryption
  - OAEP is common choice

RSA-OAEP (optimal asymmetric encryption padding)

- Provide better padding scheme than PKCS#1v1.5
- OAEP is such a padding scheme
  - r chosen randomly
  - G,H hash functions
  - $C = (X||Y)^e \bmod N$
- RSA one-wayness implies CCA security



Formalizing security: IND-CPA & IND-CCA for PKE

- Indistinguishability under chosen-plaintext attack formal notion can be adapted to PKE setting
  - Only difference: provide public key to adversary
- Semantic security [Goldwasser, Micali 1984]:
  - Can't learn any predicate over message
  - Equivalent to IND-CPA
- Neither model chosen-ciphertext attacks (like Bleichanbacher's attack)

IND-CPA(SE, $\mathcal{A}$):
(M_0,M_1) <-$ $\mathcal{A}$
(pk,sk) <-$ Kg ; b <-$ {0,1}
C <-$ Enc(pk,M_b)
b' <-$ $\mathcal{A}$(pk,C)
Return (b = b')

Formalizing security: IND-CPA & IND-CCA for PKE

- Can formalize chosen-ciphertext attack (CCA) security for PKE: IND-CCA

IND-CPA(SE, $\mathcal{A}$):
(M_0, M_1) <-$ $\mathcal{A}$
(pk,sk) <-$ Kg ; b <-$ {0,1}
C <-$ Enc(pk, M_b)
b' <-$ $\mathcal{A}$(pk,C)
Return (b = b')

Formalizing security: IND-CPA & IND-CCA for PKE

- Can formalize chosen-ciphertext attack (CCA) security for PKE: IND-CCA
- This is different than authenticity in AEAD
  - Why?
  - Anyone can encrypt: ciphertext forgeries trivial!
- Combine digital signatures (stay tuned) with PKE to achieve authenticity in asymmetric setting
- Reduction showing RSA uninvertability => OAEP is IND-CCA

IND-CCA(SE,$\mathcal{A}$):
(M_0,M_1) <-$ $\mathcal{A}$
(pk,sk) <-$ Kg ; b <-$ {0,1}
C <-$ Enc(pk,M_b)
b' <-$ $\mathcal{A}^{Dec}$(pk,C)
Return (b = b')

Dec(C')
If C' = C then Return ⊥
M <- Dec(sk,C)
Return M

TLS handshake for RSA transport

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

← CERT = (pk_s , signature over it)

C →

Pick random PMS
C <- Enc(pk,PMS)

PMS <- Dec(sk,C)

ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

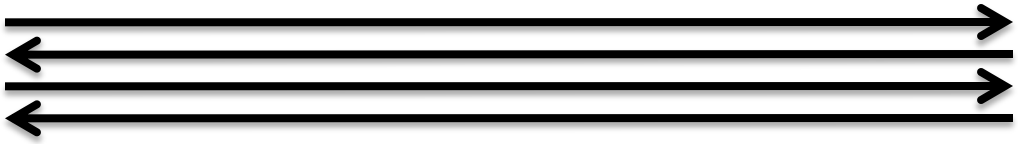
ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns)

Forward-secrecy

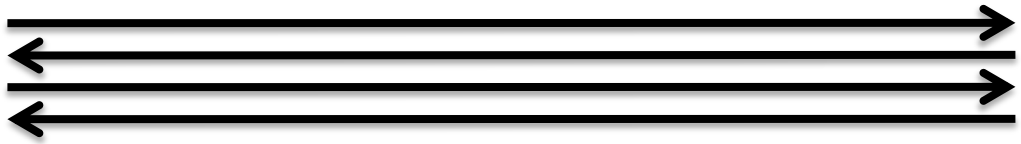
Future compromises of long-lived secrets should not enable decryption of past communications



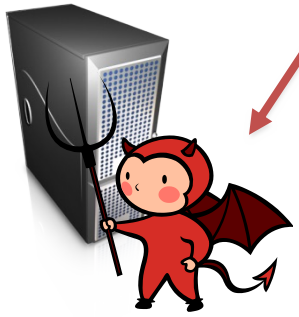
Enc(K, "Quantity: 1 , CC#: 5415431230123456")

Record encrypted transcript

Can adversary recover previous plaintext data?



Enc(K, "this definitely leaks")



Recover all long-lived secret keys

Towards achieving forward secrey

- Can’t encrypt secret session key material under long-lived encryption key
- Need ***ephemeral*** secrets that can be deleted after key exchange
  - Basic recipe: generate new asymmetric key for each key exchange
  - Could use new RSA keys, but this is pretty slow (prime generation)
- Instead: Diffie-Hellman key exchange

Diffie-Hellman using integers modulo prime

Let p be a large prime number

Fix the group $G = \mathbf{Z}_p^* = \{1,2,3,\ldots, p-1\}$

Multiplicative subgroup of finite field GF(p) (which includes 0)
So often called **finite field DH**

Then G is *cyclic*. This means one can give a member $g \in G$, called the generator, such that

$$G = \{ g^0, g^1, g^2, \ldots, g^{p-1} \}$$

General exponentiation notation for group; many of our protocols agnostic to exact group.
g^i is g^i mod p in this case

Example: p = 7. Is 2 or 3 a generator for $\mathbf{Z}_7^*$?

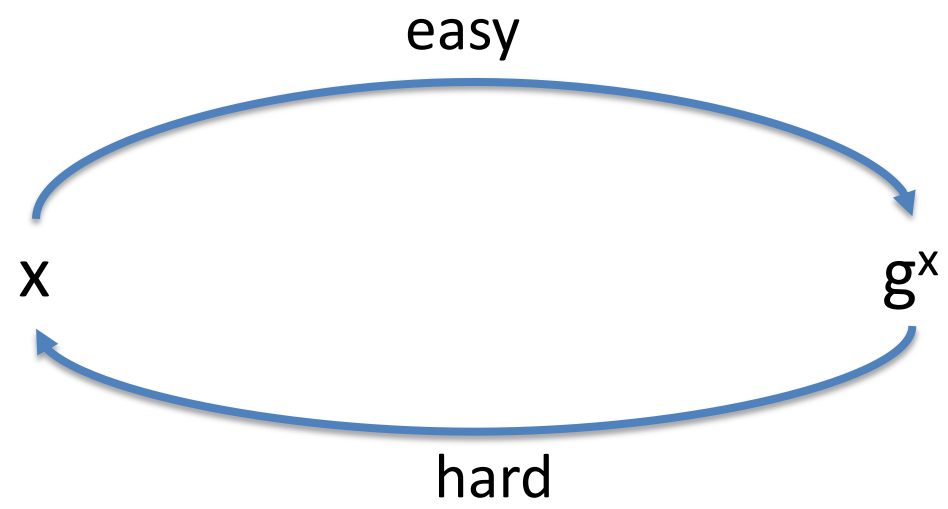
x	0	1	2	3	4	5	6
2^x mod 7	1	2	4	1	2	4	1
3^x mod 7	1	3	2	6	4	5	1

The discrete log problem

Fix a cyclic group G with generator g

Pick x at random from $\mathbf{Z}_{|G|}$

Give adversary g, $X = g^x$. Adversary's goal is to compute x



The discrete log problem

Fix a cyclic group G with generator g

Pick x at random from $\mathbf{Z}_{|G|}$

Give adversary g, $X = g^x$. Adversary's goal is to compute x

$\mathcal{A}(X)$:
for i = 2 , … , |G|-1 do
    if $X = g^i$ then
        Return i

Very slow for large groups!

$O(|G|)$

Baby-step giant-step is better:

$O(|G|^{0.5})$

Nothing faster is known for some groups.

Baby-Step Giant-Step algorithm

- DLP: Given g^x for random x, compute x

Rewrite x as $x = az + b$ with $z = \text{ceil}(p^{0.5})$

$$g^x g^{-az} = g^b$$

For b = 1, ..., z

  Store (b, g^b)

For a = 1, ..., z

  Check if $g^x g^{-az}$ equals one of precomputed g^b values

  Return az + b

- Works in time $O(p^{0.5})$ and space $O(p^{0.5})$
- Pollard rho method: reduce space to constant

Better than Baby-Step, Giant Step?

- If prime factorization of group order p-1 is “smooth”, can use Pohlig-Hellman algorithm

$$\prod_i p_i^{e_i} = \text{p-1} \qquad \text{All } p_i \text{ are primes}$$

  - Run time is $\mathcal{O}\left(\sum_i e_i(\log(p-1) + \sqrt{p_i})\right)$

- Should choose ***safe prime***: p = 2q + 1 for large prime q
- Even for safe primes: index calculus methods & NFS for DL in $\mathbf{Z}_p^*$
  - Run time is about $1.92 \cdot (\ln p)^{1/3} \cdot (\ln \ln p)^{2/3}$
  - Same as for RSA, so need pretty large p

Comparison

Security level	RSA (log N)	DLP in finite field (log p)	DLP subgroup size (log q)
80	1024	1024	1023
112	2048	2048	2047
128	3072	3072	3071
256	15360	15360	15359

Exponentiation time performance scales with $O(n^3)$ for bit length n numbers
So RSA and finite field DL are basically same performance wise

Finite field DL primes are specified in standards, see e.g.:
https://datatracker.ietf.org/doc/html/rfc7919

Unauthenticated Diffie-Hellman Key Exchange

X

Y

Pick random x from $\mathbf{Z}_{|G|}$
$X = g^x \bmod p$

Pick random y from $\mathbf{Z}_{|G|}$
$Y = g^y \bmod p$

$K = H(Y^x \bmod p)$

$K = H(X^y \bmod p)$

Get the same key. Why? $Y^x = g^{yx} = g^{xy} = X^y$

What does adversary ***need*** to compute to break security (learn K)?

What ***suffices*** for adversary to compute to break security (learn K)?

Computational Diffie-Hellman Problem

Fix a cyclic group G with generator g

Pick x,y both at random $\mathbf{Z}_{|G|}$

Give adversary g , $X = g^x$, $Y = g^y$
Adversary must compute g^{xy}

For most groups, best known algorithm solves DL of X or Y
But we have no proof that this is best approach

TLS handshake for Diffie-Hellman Key Exchange

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Check CERT using CA public verification key
Check σ

Pick random Ns

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Pick random x
$X = g^x \bmod p$

← CERT = (pk_s , signature over it)

← $p , g , X , \quad \sigma = \text{Sign}(sk_s, Nc||Ns||p||g||X)$

Pick random y
$Y = g^y \bmod p$

Y →

$PMS = g^{xy} \bmod p$

$PMS = g^{xy} \bmod p$

ChangeCipherSpec,
{ Finished, PRF(MS, "Client finished" || H(transcript)) } →

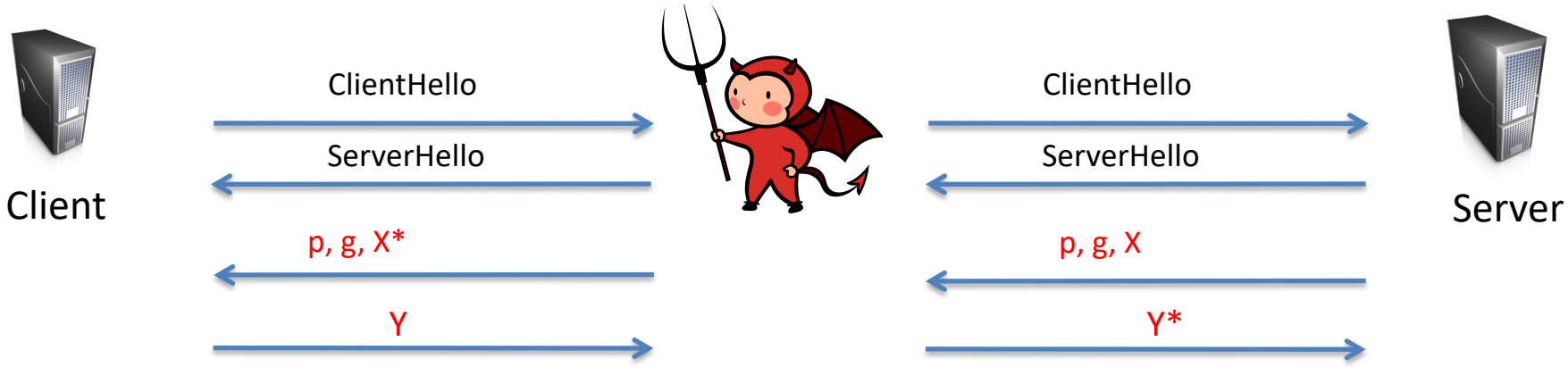
ChangeCipherSpec,
{ Finished, PRF(MS, "Server finished" || H(transcript')) } ←

Bracket notation means contents encrypted

MS <- PRF(PMS, "master secret" || Nc || Ns)

Meddler-in-the-middle (MITM) attacks

Historically called man-in-the-middle



Attacker can choose X*, Y*, so it knows discrete logs
Completes handshake on both sides
Client thinks its talking to Server
All communications decrypted by adversary, re-encrypted and forwarded to server

MITM proxy implementation:
https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/

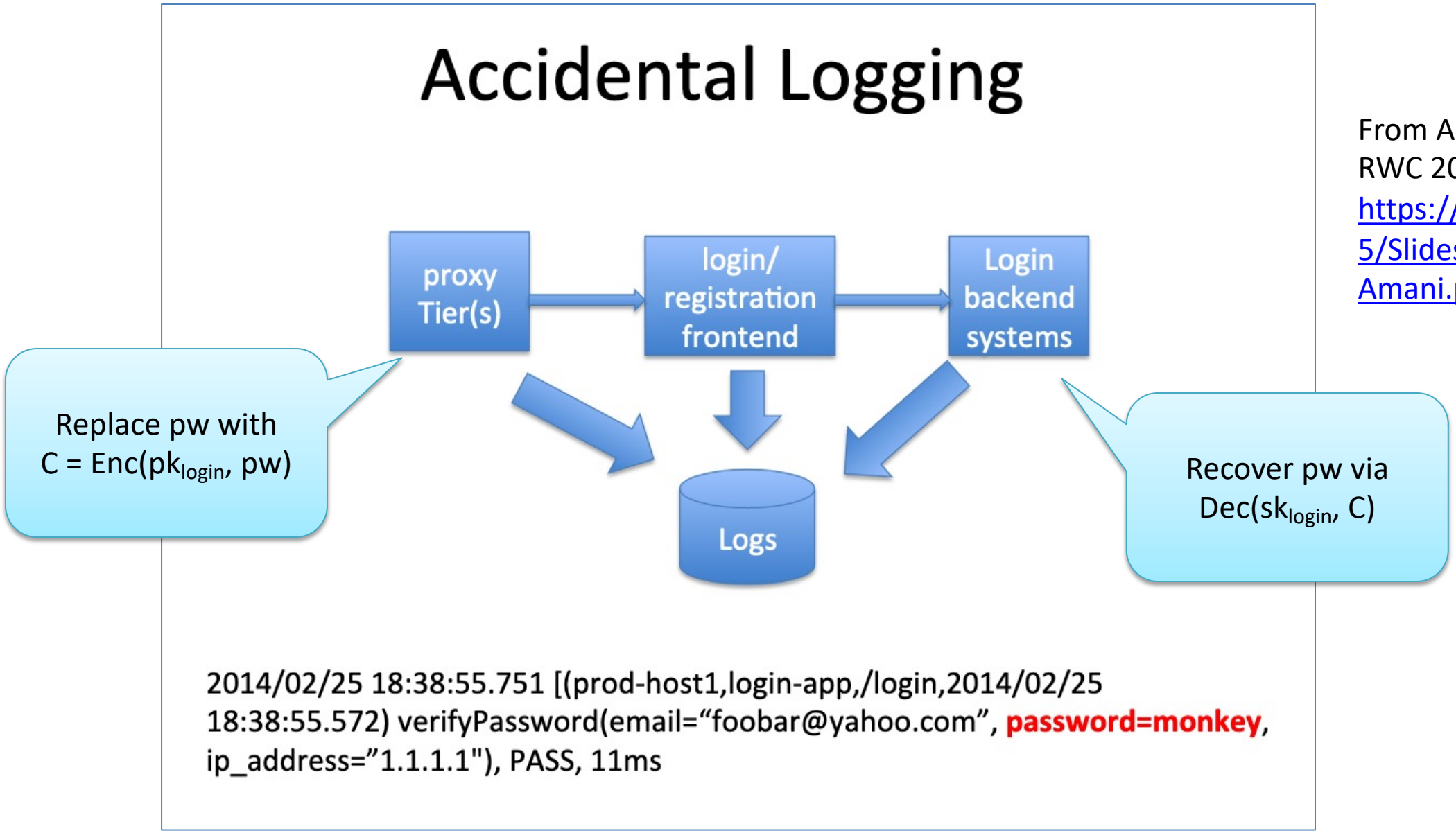
When we discuss digital signatures & PKI we will show how to prevent this

Public-key encryption beyond TLS

- Many other places where we want to use public-key encryption
  - Password encryption at TLS endpoint (LinkedIn example)

    https://rwc.iacr.org/2015/Slides/RWC-2015-Amani.pdf

The logging problem and complex infrastructure



From Amani's talk at RWC 2015 https://rwc.iacr.org/2015/Slides/RWC-2015-Amani.pdf

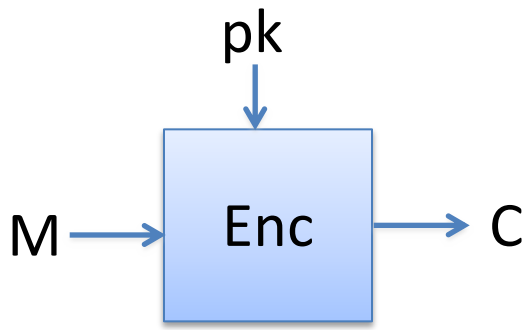
Public-key encryption beyond TLS

- Many other places where we want to use public-key encryption
  - Password encryption at TLS endpoint (LinkedIn example)
    https://rwc.iacr.org/2015/Slides/RWC-2015-Amani.pdf
  - PGP and encrypted email
  - Encrypted messaging
  - …

Public key encryption from CDH

Fix cyclic group G with generator g

Kg outputs random sk from $\mathbf{Z}_{|G|}$ and pk = g^{sk}



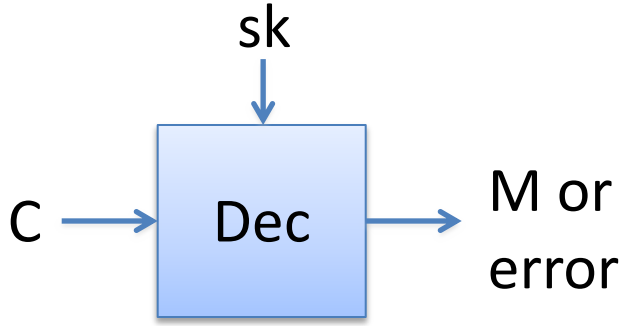
Enc(pk, M)

r <-$ $\mathbf{Z}_{|G|}$; K <- KDF(pk^r)

C_{kem} <- g^r

C_{dem} <- AEnc(K,M)

Return (C_{kem} , C_{dem})



Dec(sk, C_{kem} , C_{dem})

K <- KDF($(C_{kem})^{sk}$)

M <- ADec(K, C_{dem})

Return M

AEnc, ADec are authenticated encryption & decryption, respectively such as AES-GCM

Can optionally bind KEM ciphertext to DEM ciphertext by using it as associated data with AEnc & ADec

Example hybrid encryption schemes in practice

- ECIES (Elliptic curve with included encryption scheme), also called DHIES (Diffie-Hellman Integrated Encryption Scheme)
- libsodium / NaCL library cryptobox primitive
- Many standards: ANSI X9.63, IEEE 1363a, ISO/IEC 18033-2 and SECG SEC 1
- HPKE new standard to try to have interoperable, widely supported hybrid encryption scheme
  - https://tools.ietf.org/id/draft-barnes-cfrg-hpke-01.html

Hybrid encryption sender is *not authenticated*

- Use of AEAD does not provide *sender authenticity*
  - Anyone can generate valid ciphertext
- Does provide *non-malleability*
  - Can't modify ciphertext on unknown message M to some related M'

Enc(pk, M)
$r \leftarrow\$ \mathbf{Z}_{|G|}$; $K \leftarrow KDF(pk^r)$
$C_{kem} \leftarrow g^r$
$C_{dem} \leftarrow AEnc(K, C_{kem}, M)$
Return (C_{kem}, C_{dem})

Sender authenticity requires digital signatures, or pre-shared symmetric key (PSK)

Summary

- Diffie-Hellman can be built from integers modulo large prime
  - Sometimes called finite field Diffie-Helman
  - Should use safe primes due to Pollard-Hellman algorithm, size and efficiency about same as RSA
- Used to generate ephemeral keys for TLS key exchange, providing forward secrecy
- Used for hybrid encryption schemes, such as HPKE