Good/Bad Crypto (cont.) & Bitcoin

Announcements!

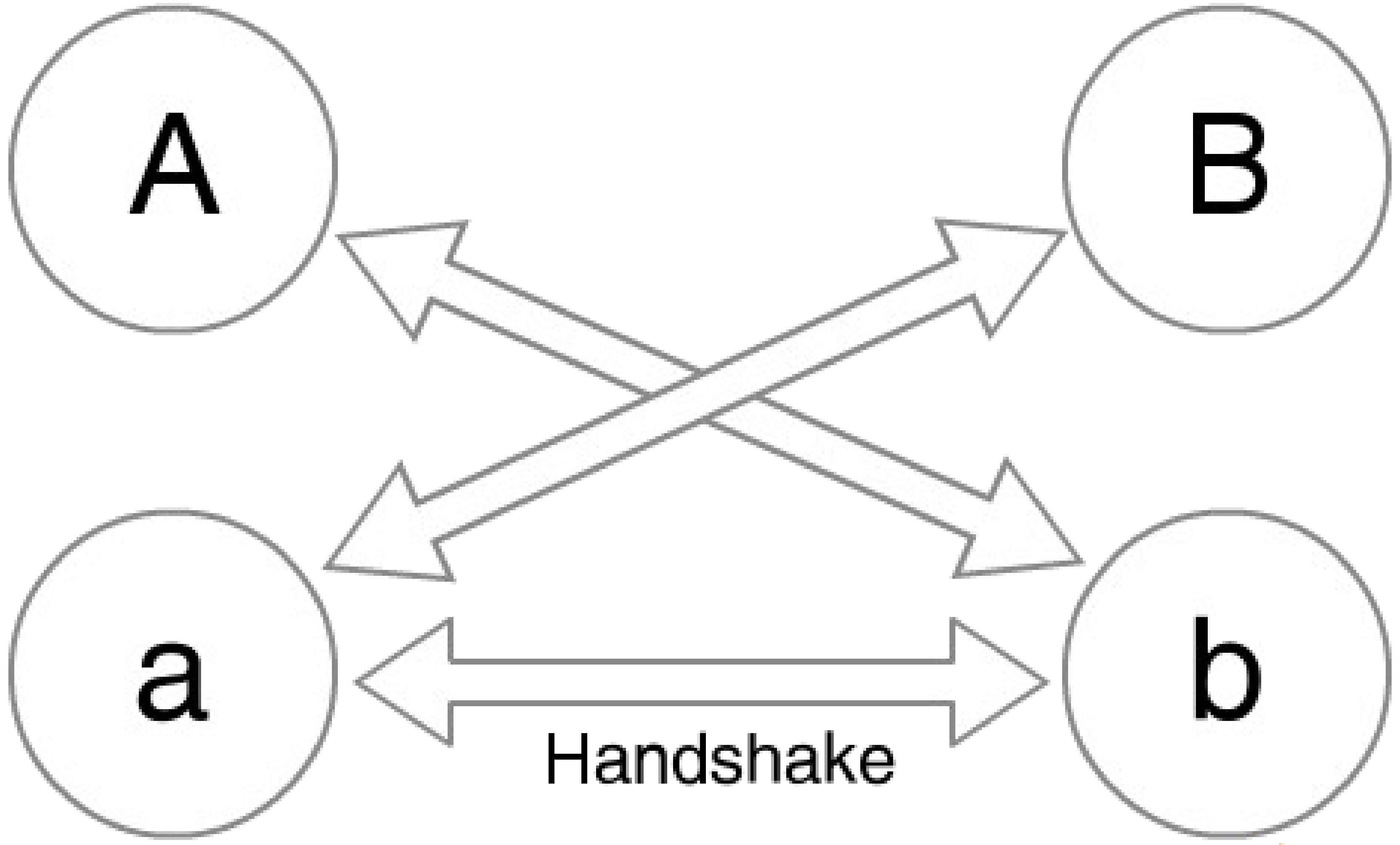
- Midterm 1 Monday, 5-7 pm
  - Bring your student ID
- Project 1 due tonight
  - Make only 1 submission per group!

Exercise:
Send me an encrypted message

- Make sure no one else can read the message
  - Use any communication method you want
- How can you find my public key?
  - How can you be sure it's me?
  - How can I be sure it's you?
- How can I respond in encrypted form?
- Does the communication have forward secrecy?
- Does it have integrity? Authentication?
- Is it deniable? Or non-repudiable?

Signal
Authenticated Diffie-Hellman with Deniability

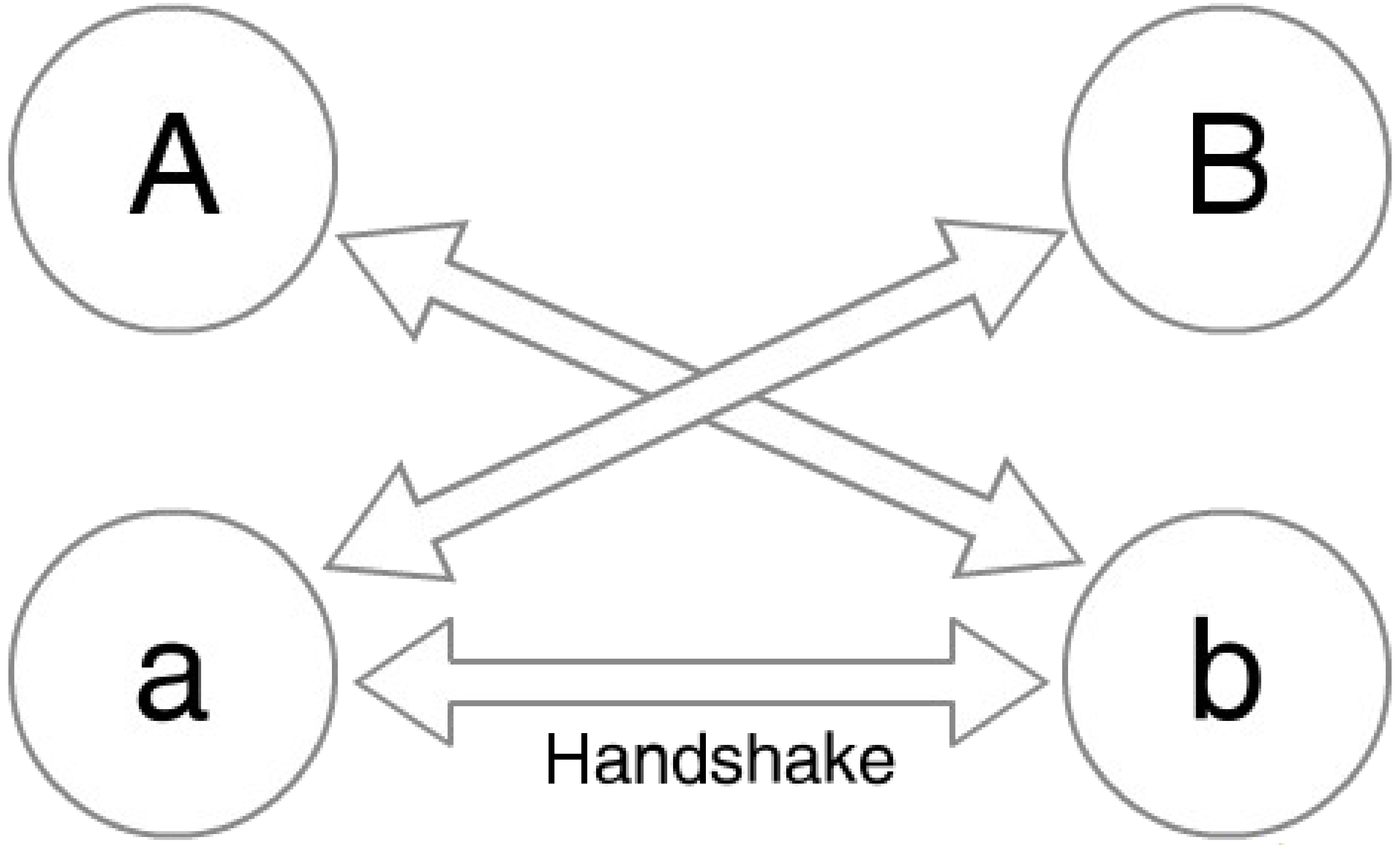
- Alice has long term secret key A, generates ephemeral secret key a.
- Bob has long term secret key B, generates ephemeral secret key b.



Signal
Authenticated Diffie-Hellman with Deniability

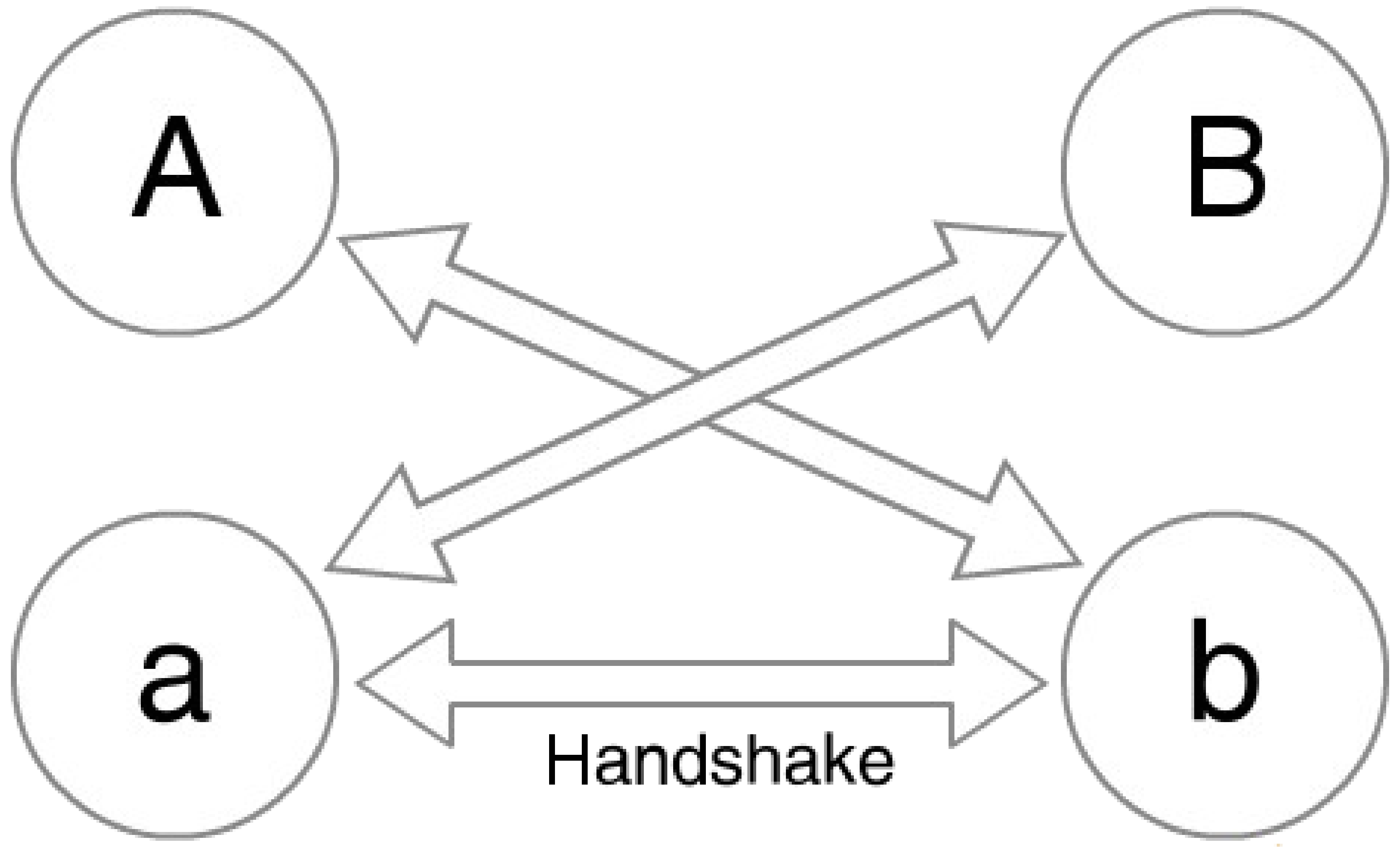
Alice sends $\mathbf{g}^{\mathbf{A}}$ and $\mathbf{g}^{\mathbf{a}}$

Bob sends $\mathbf{g}^{\mathbf{B}}$ and $\mathbf{g}^{\mathbf{b}}$

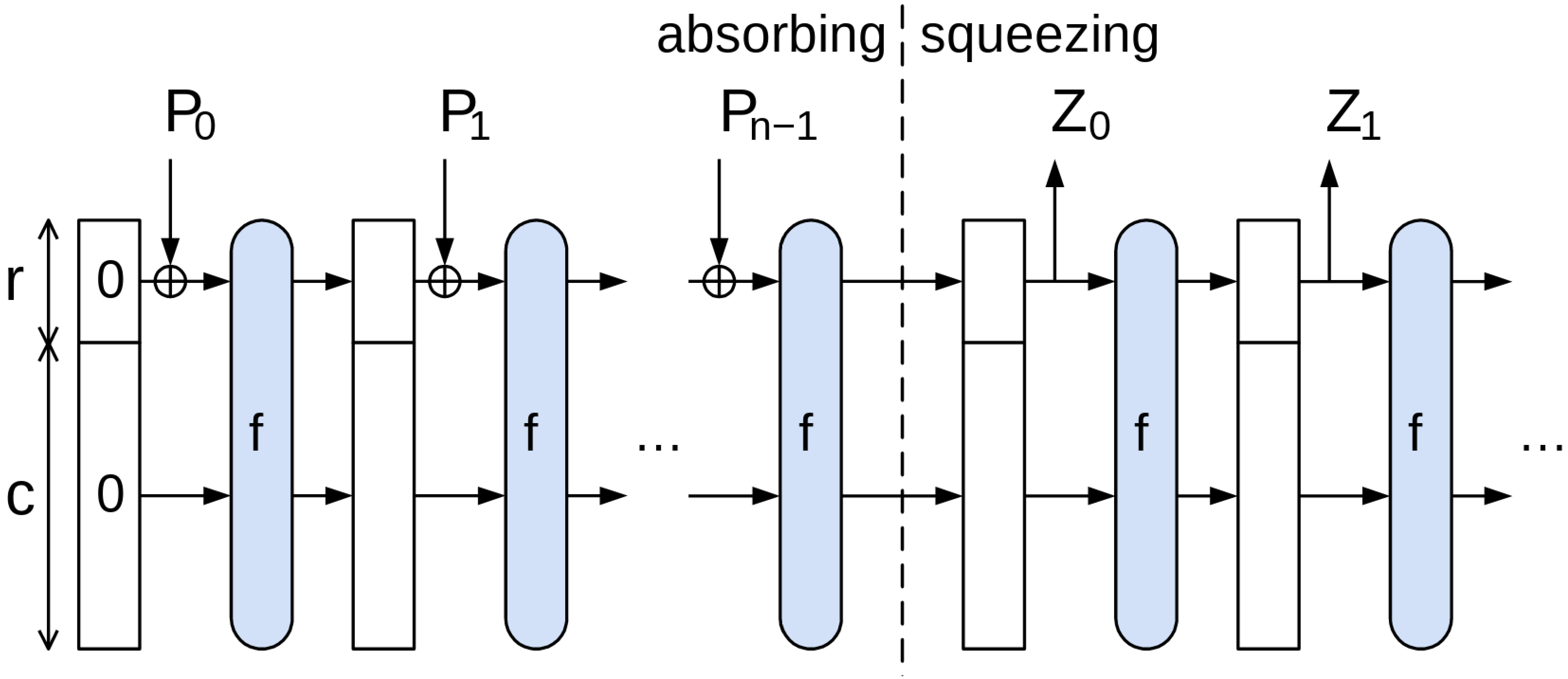


Signal
Authenticated Diffie-Hellman with Deniability

Both compute **KDF(g^{Ab}, g^{aB}, g^{ab})**



SHA3 (Keccak) Cryptographic Sponge Construction



Unusability: PGP

- I ***hate*** Pretty Good Privacy
  - But not because of the cryptography...
- The PGP cryptography is decent...
  - Except it lacks "Forward Secrecy":
    If I can get someone's private key I can decrypt all their old messages
- The metadata is awful...
  - By default, PGP says who every message is from and to
    - It makes it much faster to decrypt
  - It is hard to hide metadata well, but its easy to do things better than what PGP does
- It is never transparent
  - Even with a "good" client like GPG-tools on the Mac
  - And I don't have a client on my cellphone

Unusability: How do you find someone's PGP key?

- Go to their personal website?
- Check their personal email?
- Ask them to mail it to you
  - In an unencrypted channel?
- Check on the MIT keyserver?
  - And get the old key that was mistakenly uploaded and can never be removed?

Search results for 'nweaver icsi edu berkeley'

Type	bits/keyID	Date	User ID
pub	4096R/8A46A420	2013-06-20	Nicholas Weaver <nweaver@icsi.berkeley.edu> Nicholas Weaver <n_weaver@mac.com> Nicholas Weaver <nweaver@gmail.com>
pub	2048R/442CF948	2013-06-20	Nicholas Weaver <nweaver@icsi.berkeley.edu>

Unusability:
openssl libcrypto and libssl

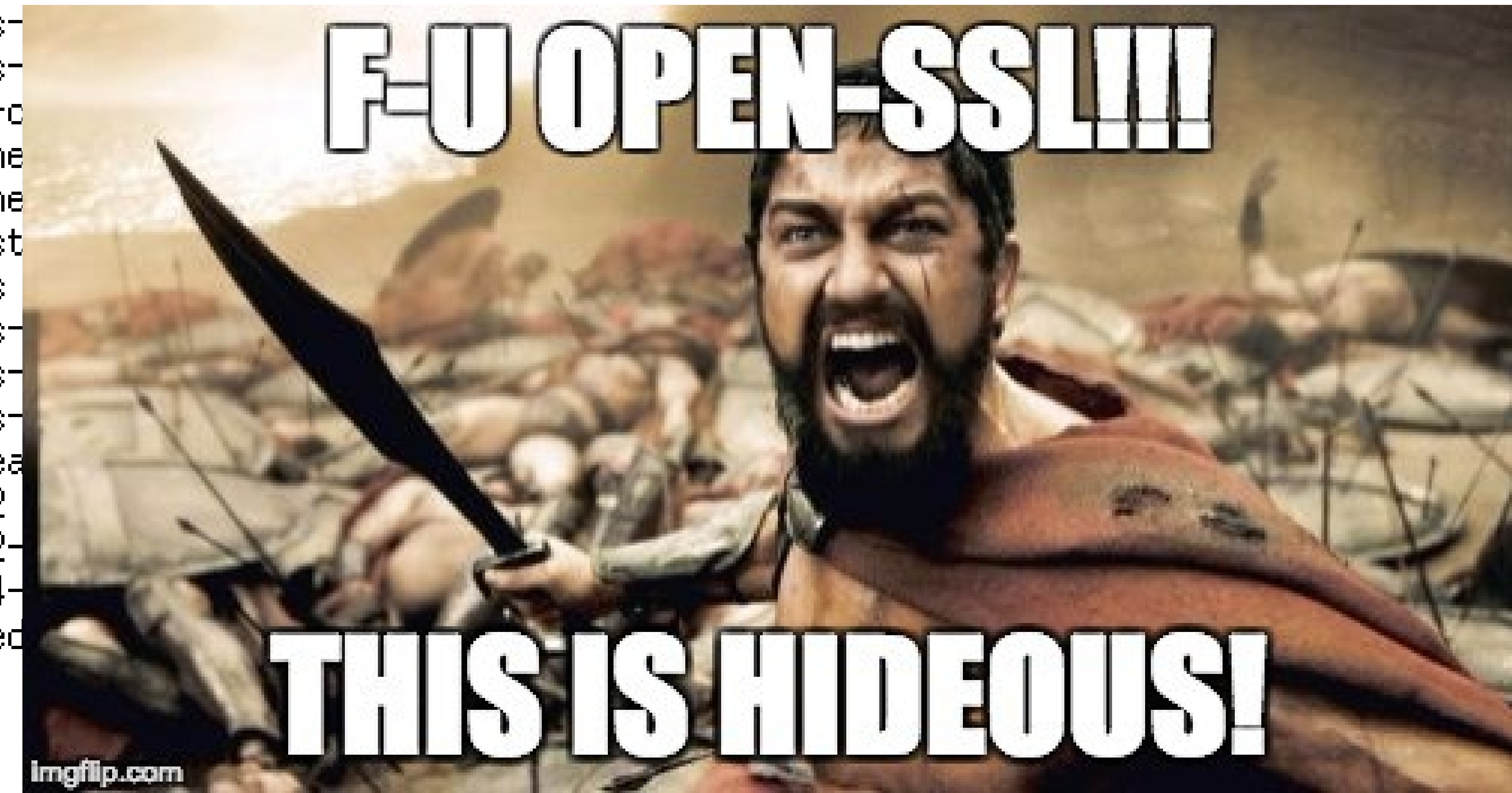
- OpenSSL is a nightmare...
  - A gazillion different little functions needed to do anything
- So much of a nightmare that I'm not going to bother learning it to teach you how bad it is
  - This is why the old python-based project didn't give this raw even though it used OpenSSL under the hood
- But just to give you an idea:
  The command line OpenSSL utility options:

```
OpenSSL> help
openssl:Error: 'help' is an invalid command.

Standard commands
asn1parse         ca                ciphers           cms
crl               crl2pkcs7         dgst              dh
dhparam           dsa               dsaparam          ec
ecparam           enc               engine            errstr
gendh             gendsa            genpkey           genrsa
nseq              ocsp              passwd            pkcs12
pkcs7             pkcs8             pkey              pkeyparam
pkeyutl           prime             rand              req
rsa               rsautl            s_client          s_server
s_time            sess_id           smime             speed
spkac             srp               ts                verify
version           x509

Message Digest commands (see the `dgst' command for more details)
md4               md5               mdc2              rmd160
sha               sha1

Cipher commands (see the `enc' command for more details)
aes-
aes-
bf-c
came
came
cast
des
des-
des-
des-
idea
rc2
rc2-
rc4-
seed
```



An Old Cryptofail:
Too Short Keys

- During WWII, the Germans used ***enigma***:
  - System was a "rotor machine": A series of rotors, with each rotor permuting the alphabet and every keypress incrementing the settings
    - Key was the selection of rotors, initial settings, and a permutation plugboard
    - Which is not all that much entropy!
- The British built a system (the "Bombe") to brute-force Enigma
  - Required a known-plaintext (a "crib") to verify decryption: e.g. the weather report
  - Sometimes the brits would deliberately "seed" a naval minefield for a chosen-plaintext attack

Another Cryptofail: Two-Time Pad

- What if we reuse a key K jeeeest once in a One Time Pad?
- Alice sends **C = E(M, K)** and **C' = E(M', K)**
- Eve observes **M ⊕ K** and **M' ⊕ K**
  - Can she learn anything about M and/or M' ?
- Eve computes **C ⊕ C' = (M ⊕ K) ⊕ (M' ⊕ K)**
  = (M ⊕ M') ⊕ (K ⊕ K)
  = (M ⊕ M') ⊕ 0
  = M ⊕ M'
- Now she knows which bits in **M** match bits i
- And if Eve already knew **M**, now she knows
  - Even if not, Eve can guess **M** and ensure that **M'** is consi

VENONA:
Pad Reuse in the Real World

- The Soviets used one-time pads for communication from their spies in the US
  - After all, it is provably secure!
- During WWII, the Soviets started reusing key material
  - Uncertain whether it was just the cost of generating pads or what...
- VENONA was a US cryptanalysis project designed to break these messages
  - Included confirming/identifying the spies targeting the US Manhattan project
  - Project continued until 1980!
- ***Not declassified until 1995!***
  - So secret even President Truman wasn't informed about it.
  - But the Soviets found out about it in 1949, but their one-time pad reuse was fixed after 1948 anyway

2-Time Pad Cryptofail Remarkably Common

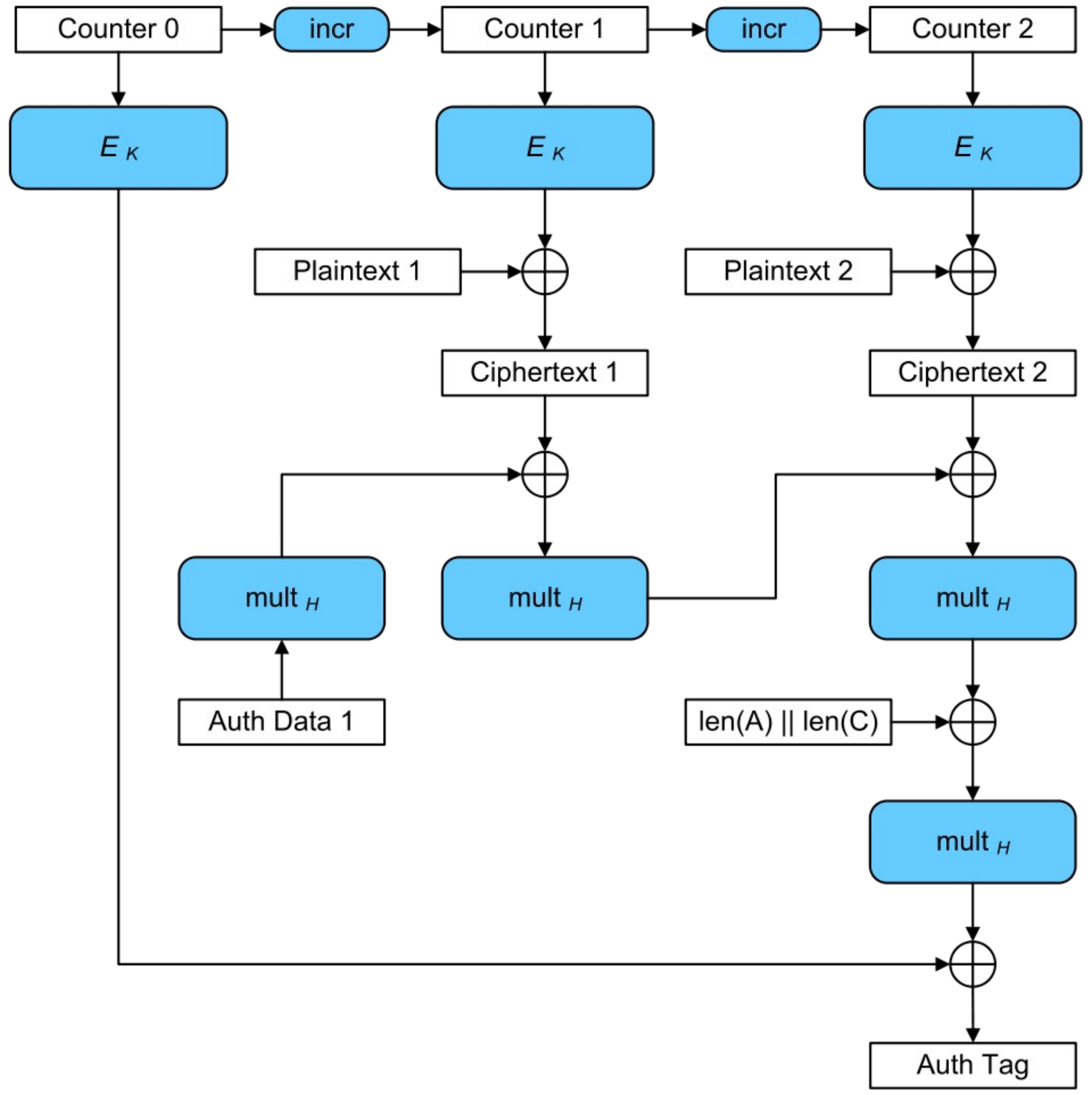
- Actually happens more often than you'd like...
  - Because if you use CTR mode and repeat the IV, you are doing the same thing!
- Recently discovered in WiFi implementations!
  - WiFi breaks up the message into a series of packets, each packet is encrypted separately

Cryptofail Hotness: KRACK attack...

- To actually encrypt the individual packets: IV of a packet is {Agreed IV || packet counter}
  - Thus for each packet you only need to send the packet counter (48 bits) rather than the full IV (128b)
- Multiple different modes
  - One common one is CCM (Counter with CBC-MAC)
    - MAC the data with CBC-MAC
      Then encrypt with CTR mode
  - The highest performance is GCM (Galois/Counter Mode)
- KRACK:
  - "Hey WiFi Device, reset your packet counter"
    "Okeydoke"
- But if you thought CTR mode was bad on IV reuse...
  - GCM is worse: A couple of reused IVs can reveal enough information to forge the authentication!
- Discovered a year and a half ago, fairly quickly patch, but still!

GCM...

- GCM is like CTR mode with a twist...
  - The confidentiality is pure CTR mode
  - The "Galois" part is a hash of the cipher text
    - The only secret part being the "Auth Data"
- Reuse the IV, what happens?
  - Not ***only*** do you have CTR mode loss of confidentiality...
  - But if you do it enough, you lose confidentiality on the Auth Data...
  - So you lose the integrity that GCM supposedly provided!



DSA Signatures...

- Based on Diffie-Hellman
  - Two initial parameters, **L** and **N**, and a hash function **H**
    - **L** == key length, eg 2048
      N <= len(H), e.g. 256
    - An N-bit prime **q**, an L-bit prime **p** such that **p - 1** is a multiple of **q**, and
      $\mathbf{g = h^{(p-1)/q} \bmod p}$ for some arbitrary **h** $(1 < h < p - 1)$
    - {**p, q, g**} are public parameters
- Alice creates her own random private key **x < q**
  - Public key $\mathbf{y = g^x \bmod p}$

Alice's Signature...

- Create a random value $\mathbf{k} < \mathbf{q}$
  - Calculate $\mathbf{r = (g^k \bmod p) \bmod q}$
    - If $\mathbf{r} = 0$, start again
  - Calculate $\mathbf{s = k^{-1} (H(m) + xr) \bmod q}$
    - If $\mathbf{s} = 0$, start again
  - Signature is {**r**, **s**} (Advantage over an El-Gamal signature variation: Smaller signatures)
- Verification
  - $\mathbf{w = s^{-1} \bmod q}$
  - $\mathbf{u_1 = H(m) * w \bmod q}$
  - $\mathbf{u_2 = r * w \bmod q}$
  - $\mathbf{v = (g^{u_1} y^{u_2} \bmod p) \bmod q}$
  - Validate that $\mathbf{v = r}$

But Easy To Screw Up…

- **k** is not just a nonce… It must be random and ***secret***
  - If you know **k**, you can calculate **x**
- And even if you just reuse a random **k**… for two signatures sa and sb
  - A bit of algebra proves that $\mathbf{k = (H_A - H_B) / (}$
- A good reference:
  - How knowing k tells you x: https://rdist.root.org/2009/05/17/the-debian-pgp-disaster-that-almos
  - How two signatures tells you k: https://rdist.root.org/2010/11/19/dsa-requirements-for-random-k-v

And ***NOT*** theoretical: Sony Playstation 3 DRM

- The PS3 was designed to only run ***signed*** code
  - They used ECDSA as the signature algorithm
  - This prevents unauthorized code from running
  - They had an ***option*** to run alternate operating systems (Linux) that they then removed
- Of course this was catnip to reverse engineers
  - Best way to get people interested: ***remove*** Linux from a device...
- It turns for out one of the key authentication keys used to sign the firmware...
  - Ended up reusing the same k for multiple signatures!

And ***NOT*** Theoretical: Android RNG Bug + Bitcoin

- OS Vulnerability in 2013
  Android "SecureRandom" wasn't actually secure!
  - Not only was it low entropy, it would occasionally return the ***same value multiple times***
- Multiple Bitcoin wallet apps on Android were affected
  - "Pay B Bitcoin to Bob" is signed by Alice's public key using ECDSA
    - Message is broadcast publicly for all to see
  - So you'd have cases where "Pay B to Bob" and "Pay C to Carol" were signed with the same **k**
- So ***of course*** someone scanned for ***all*** such Bitcoin transactions