CS 1699 Privacy in the Electronic Society

William Garrison
bill@cs.pitt.edu
6311 Sennott Square
http://cs.pitt.edu/~bill/1699

05: Limits of cryptography

Today's topics: Why isn’t crypto enough?

Key servers can be exploited (including by their owners!)

- Is iMessage private?

Cryptographic primitives used naively can be harmful

- Padding is needed to prevent homomorphic attacks

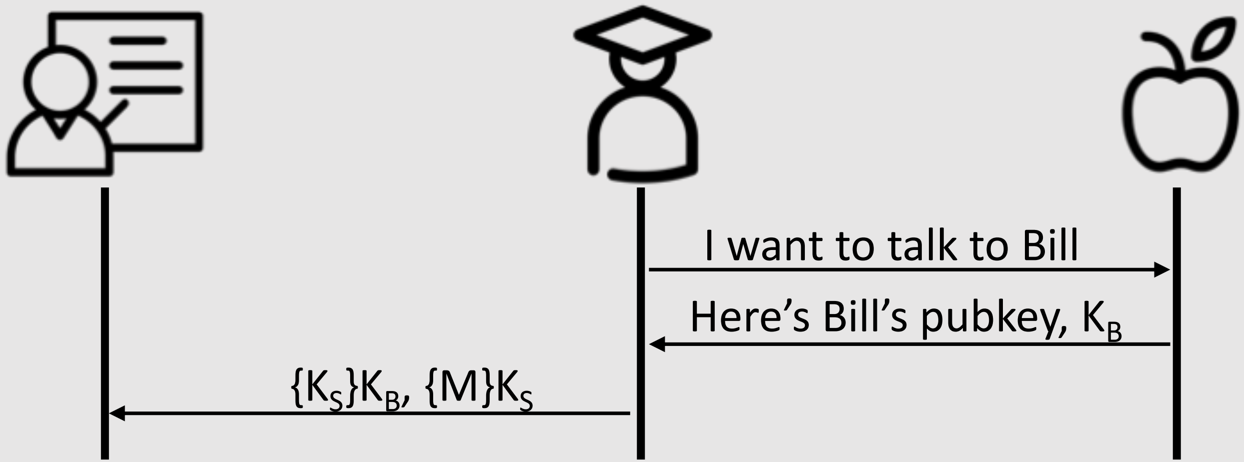
Brute-force attacks, like all other attacks, only get better

- EFF's Deep Crack (and later Distributed.net) break 56-bit DES

Random numbers are important, and easy to get wrong

- Netscape, Kerberos, Sony PS3…

Is Apple's iMessage private?



Note: This protocol is not quite this simple. We'll discuss it again later in the term in more detail.

Should I trust Apple to distribute keys?

- What if they lie? Make a mistake? Government requests?

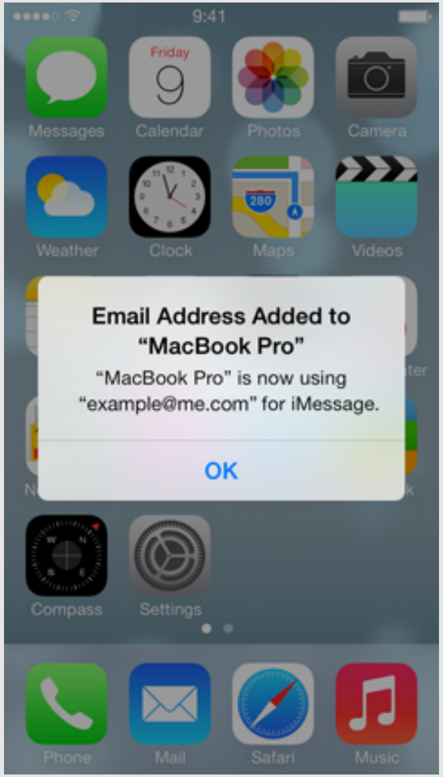
Further complications from multi-device

You might want to chat on both your iPhone and Mac

- How can we transfer private keys?
- Instead, key per device; key server sends several keys
- Hybrid crypto means this isn't too expensive
  - $\{M\}K_S$, $\{K_S\}K_{B1}$, $\{K_S\}K_{B2}$…
- How might this change the trust I have in the key server?

Luckily, Apple alerts you if a key was added for your account

- But will they always? Mistakes, lies, government…



Recently, Apple announced big changes in China

Starting Feb 28th, iCloud backups for users specifying China as their country will be stored by GCBD in China

- Improved performance for Chinese users!
- … But also satisfying new regulations on cloud services
- This data is encrypted, so no problem… ?

Mud puddle test: Someone has the keys!

- If you fell in a mud puddle, destroying your phone and suffering memory loss, could you get your data back? (Yes)
- With separation of duties, this might be okay
  - In the US, Apple stores keys while outsourcing bulk data storage
- So who stores the keys in the new arrangement? How vulnerable are they?

Cryptography is subtle and easy to misuse

Recall that signing and decrypting are the same in (many) public-key cryptosystems

- If I am willing to sign messages, I may be a decryption oracle

Assume I run a service to sign homework submissions to prove they were done on time

- My public key is (n, e), my private key is d
- You send M, I send you back $M^d \bmod n$
- Now let's say you discover some $C = M^e \bmod n$
  - What is this? What if you send it to me for signing?
  - Can I prevent this?

But it gets worse...

Instead of sending C, you could be more clever. Let:

- R be some random junk number
- $X = R^e \bmod n$
- $Y = XC \bmod n$

Send me Y to sign instead:

- $R^{-1}Y^d \bmod n = R^{-1}(XC)^d \bmod n$
- $= R^{-1}X^dC^d \bmod n$
- $= R^{-1}R^{ed}M^{ed} \bmod n$
- $= R^{-1}RM \bmod n$
- $= M \bmod n$

But I wouldn't recognize Y^d as being meaningful!

Why does this work?

RSA has multiplicative homomorphism

- $(A^e \bmod n)(B^e \bmod n) = (AB)^e \bmod n$
- $E(x)E(y) = E(xy)$

But this only works if encrypting/signing raw data

- In practice, RSA should not be used in this way
- Instead, padding functions such as OAEP / PKCS#1 randomly pad the message
  - M to $P(M)$
  - $P^{-1}\big(P(M)\big) = M, P^{-1}\left(D\left(E\big(P(M)\big)\right)\right) = M$
  - But, $P^{-1}\big(P(A)P(B)\big) \neq AB$

Beyond this, we should avoid services that are decryption oracles

- Never use the same key for two purposes
- Here, use different keys for signing and secure messaging

Sometimes, straightforward attacks become feasible in time

DES was a symmetric cipher developed in the early 70s

- Federal standard in Nov 1976

DES's biggest criticism was the (short) 56-bit key size

- 72,057,594,037,927,936 possible keys
- But, US gov't continued to stand by the infeasibility of an attack

In 1998, EFF built a $250k specialized, parallel machine to crack DES

- 29 circuit boards, 64 custom chips per board
- These 1,856 chips could test 90 billion keys per second
- Cracked key in 56 hours

Super manageable for govmts, b

In 2002, EFF and Distributed.net paired up to use 100,000 volunteer PCs

- These general-purpose machines cracked DES in 22.25 hours

What about randomness as an input to crypto?

Bad randomness can mean the cryptography is useless

- Let's say I encrypt data with a random key generated with a secure PRNG seeded by rolling a fair 6-sided die
  - What could go wrong? What can an attacker do, and how will it hurt me? Then you only need to try every seed (only 6!) instead of breaking the keyspace.
- What if I roll the die 100 times and append? Add? XOR?

Append: then you have 6^100 options... not

Add: leaves you in the (very small) r

XOR: obviously bad(

Misuse of a PRNG can have huge implications for crypto

- Predictable keys or IVs
- Padding no longer secure
- Some ciphers require random nonce in addition to keys (think something like an IV)

Ian Goldberg and David Wagner discovered such a problem in Netscape's key generation

```
global variable seed;
RNG_CreateContext()
      /* Time elapsed since 1970 */
      (seconds, microseconds) = time of day;
      pid = process ID;
      ppid = parent process ID;
      a = mklcpr(microseconds);
      b = mklcpr(pid + seconds + (ppid << 12));
      seed = MD5(a, b);

/* not cryptographically significant; shown for
completeness */
mklcpr(x)
      return ((0xDEECE66D * x + 0x2BBB62DC) >> 1);
```

So what's wrong with this?

The PRNG used was considered secure, but it was seeded using a few basic values:

- Current time (seconds and milliseconds)
- Process ID (PID, 15 bits)
- Parent process ID (PPID, 15 bits)

Issues?

- Can any of these be observed if logged in to the same machine? Basically (pretty c
  - How will the attacker know if they're right? Challenge is randomly generated also, then sent by the browser in p
- What other tricks could we use to narrow the choices?
- How many bits of security is this, even if we aren't co-located?
  - a = mklcpr(microseconds);
  - b = mklcpr(pid + seconds + (ppid << 12)); Pretty easy to guess the second (since we're communicating); also, once up

Kerberos v4 suffered a similar bug

Kerberos is an open-source network authentication protocol

- Widely used in academia, default in Windows, etc.
  - Used in our department for AFS distributed filesystem
- We’ll discuss authentication later in the semester
- Closely-studied, open-source

PRNG used to generate session keys was seeded with several 32-bit values XOR'd together:

- Time in seconds and milliseconds
- PID of server
- Cumulative count of session keys generated since launch
- Host ID of the machine

We're only XOR-ing them!! Still only a 32-bit space

How many bits of randomness is this?

See next page: the randomness is constrained not by th

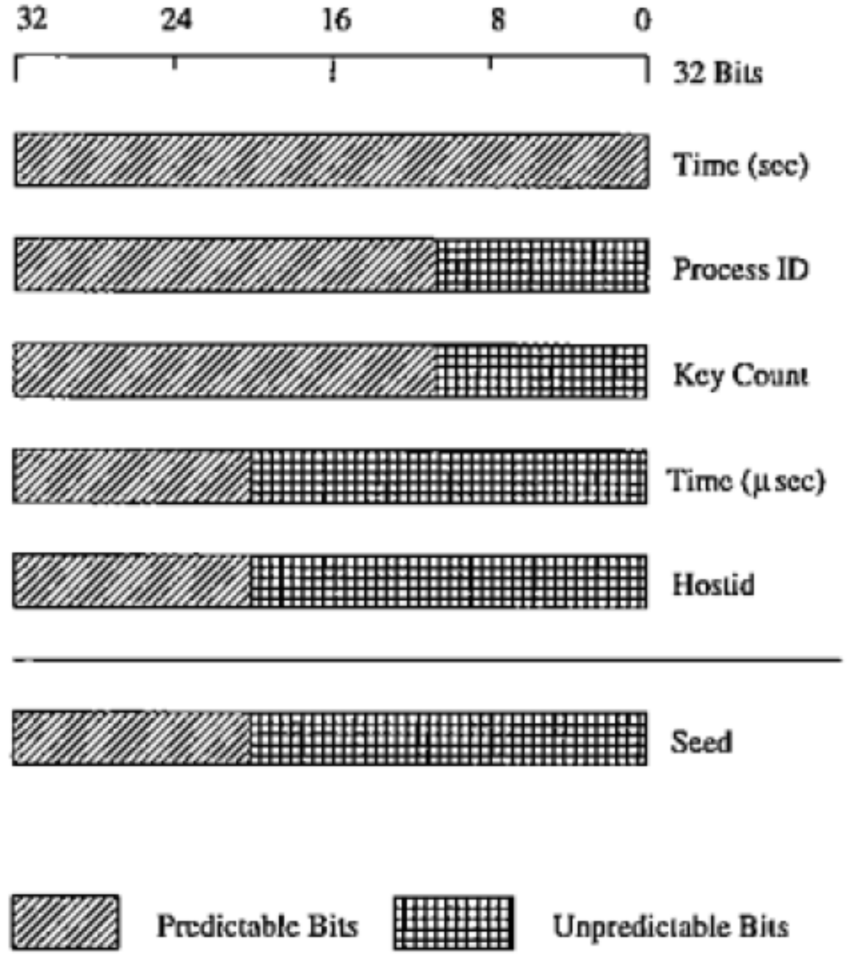


Figure 2. Random Number Generator Seed

A related mistake by Sony allowed PS3 protections to be bypassed entirely

Sony's goal: PS3 should not install any software except that provided by Sony

- Sign firmware updates, public key used to verify before install
- Where did the public key come from? Shipped with the original firmware. Update with new one later if needed, sig
- What would happen if the corresponding private key were discovered? Then you can rewrite the firmware and sign the modified version, not update the new keypairs, etc. The whole thing breaks.

ECDSA (Elliptic Curve Digital Signature Algorithm)

- $e = \text{HASH}(\text{message}),\ k = \text{ random value},\ d = \text{ private key}$
- Signature is (R, S), where:
  - R is a function of k (and public values)
  - $S = \frac{e+dR}{k}$
- k must be random and fresh!

What happens if m is reused?

R is a function of k (and constants), so $R_1 = R_2$

$$S_1 = \frac{e_1 + dR}{k}, S_2 = \frac{e_2 + dR}{k}$$

So, $S_1 - S_2 = \frac{e_1 - e_2}{k}$, and $k = \frac{e_1 - e_2}{S_1 - S_2}$

So, an attacker can find k given 2 signatures using the same key, d (and same k)

S is part of the signature, and e is the hash we're validating against, so both are known

$$d = \frac{kS_1 - e_1}{R}$$

Given k, we can find the private key d!

So: Don't use the same random value k twice!

Guess what Sony did…

Yep

Conclusions

Cryptography isn't enough to protect our privacy alone

- Central authorities can lie, make mistakes, or be coerced
  - Trust is relative!
- Primitives can be misused
  - Don't use the same key for 2 purposes!
- Attacks get better over time (threat models change)
- Randomness can be difficult True randomness is hard to come by
- Know the proper usage and up-to-date best practices for the algorithms you use Think!

Next: Side-channel attacks