WILLIAM & MARY
CHARTERED 1693

CSCI 667: Concepts of Computer Security

Lecture 5

Prof. Adwait Nadkarni

Derived from slides by William Enck, Micah Sherr, Patrick McDaniel and Peng Ning

Using hashes as authenticators

- Consider the following scenario
  - Prof. Smart E. Pants has not decided if she will cancel the next lecture.
  - When she does decide, she communicates to Bob the student through Mallory, her evil TA.
  - She does not care if Bob shows up to a cancelled class, but she does not want students to not show up if the class hasn't been cancelled
  - Prof. Pants does not trust Mallory to deliver the message.
- Prof. Smart E. Pants and Bob use the following protocol:
  - Prof. Pants invents a secret t
  - Prof. Pants gives Bob h(t), where h() is a crypto hash function
  - If she cancels class, she gives t to Mallory to give to Bob
  - If does not cancel class, she does nothing
  - If Bob receives the token t, he knows that Prof. Pants sent it

Hash Authenticators

- Why is this protocol secure?
  - t acts as an authenticated value (authenticator) because Mallory could not have produced t without inverting h()
  - Note: Mallory can convince Bob that class is occurring when it is not by simply not delivering t (but we assume Bob is smart enough to come to that conclusion when the room is empty)
- Note that it is important that Bob gets the original value h(t) from Alice directly (was provably authentic)

Hash chain

- Now, consider the case where Alice wants to do the same protocol, only for all 26 classes (the semester)
- Alice and Bob use the following protocol:

1. Alice invents a secret t
2. Alice gives Bob $H^{26}(t)$, where $H^{26}()$ is 26 repeated uses of $H()$.
3. If she cancels class on day d, she gives $H^{(26-D)}(t)$ to Mallory, e.g.,

   If cancels on day 1, she gives Mallory $H^{25}(t)$

   If cancels on day 2, she gives Mallory $H^{24}(t)$

   …….

   If cancels on day 25, she gives Mallory $H^{1}(t)$

   If cancels on day 26, she gives Mallory t

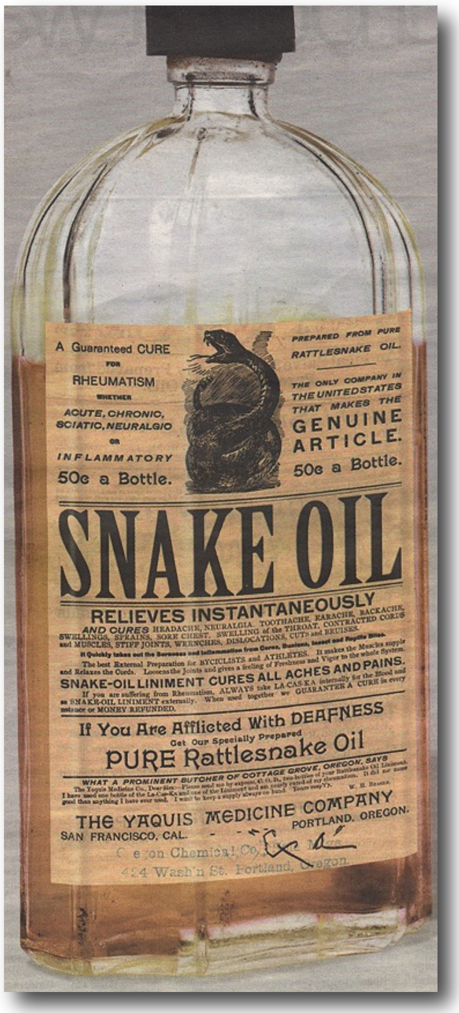
4. If Alice does not cancel class, she does nothing

– If Bob receives the token t, he knows that Alice sent it

Hash Chain (cont.)

- Why is this protocol secure?
  - On day d, $H^{(26-d)}(t)$ acts as an authenticated value (authenticator) because Mallory could not create t without inverting $H()$ because for any $H^k(t)$ she has $k>(26-d)$
  - That is, Mallory potentially has access to the hash values for all days prior to today, but that provides no information on today's value, as they are all post-images of today's value
  - Note: Mallory can again convince Bob that class is occurring by not delivering $H^{(26-d)}(t)$
  - Chain of hash values are ordered authenticators
- Important that Bob got the original value $H^{26}(t)$ from Alice directly (was provably authentic)

Basic truths of cryptography



- Cryptography is not frequently the source of security problems
  - Algorithms are well known and widely studied
  - Vetted through crypto community
  - Avoid any “proprietary” encryption
  - Claims of “new technology” or “perfect security” are almost assuredly **snake oil**

Building systems with cryptography

- Use quality libraries
  - SSLeay, cryptolib, openssl
  - Find out what cryptographers think of a package before using it
- Code review like crazy
- Educate yourself on how to use library
  - Understand caveats by original designer and programmer

Cipher.getInstance("*AES*") defaults to ECB mode!

Common pitfalls

- Generating randomness
- Storage of secret keys
- Virtual memory (pages secrets onto disk)
- Protocol interactions
- Poor user interface
- Poor choice of parameters or modes

Let's review...

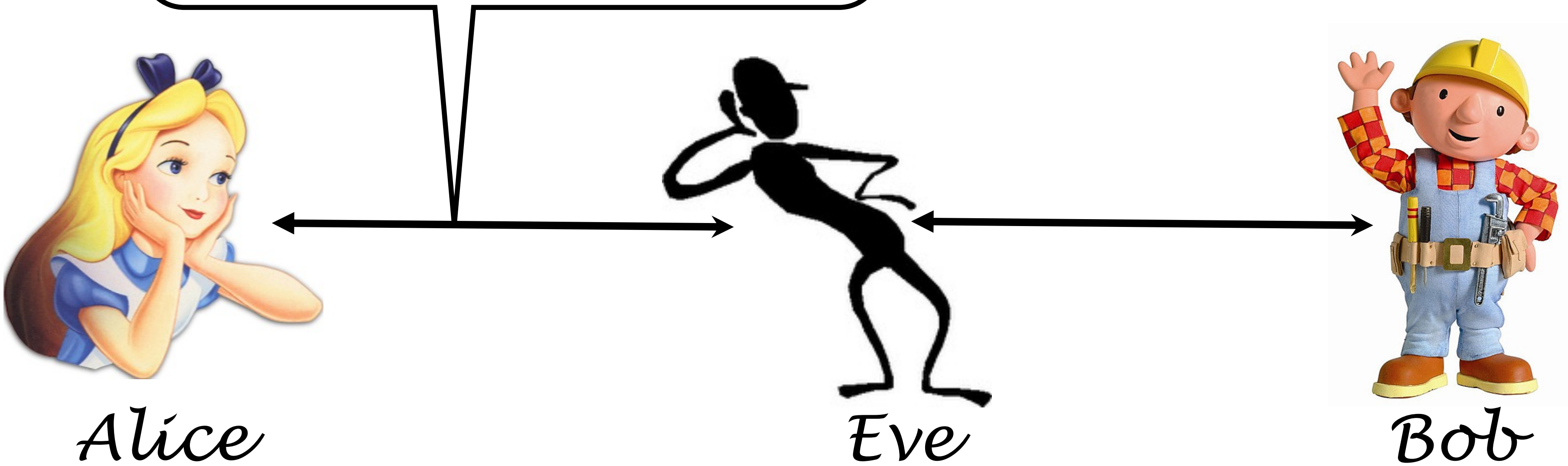
Private-key crypto is like a door lock

Why?

Encryption and Message Authenticity

Src = Alice, Dest = Bob
Msg = E_{k1}{{"network security is fun",
MAC_{k2}("network security is fun!")}}

What's the hard part?



Without knowing *k1*, Eve can't read Alice's message.

Without knowing *k2*, Eve can't compute a valid MAC for her forged message.

Public Key Crypto
(10,000 ft view)

- Separate keys for encryption and decryption
  - Public key: anyone can know this
  - Private key: kept confidential
- Anyone can encrypt a message to you using your public key
- The private key (kept confidential) is required to decrypt the communication
- Alice and Bob no longer have to have *a priori* shared a secret key

Public Key Cryptography

- Each key pair consists of a public and private component: k^+ (public key), k^- (private key)

$$D_{k^-}(E_{k^+}(m)) = m$$

- Public keys are distributed (typically) through public key certificates
  - Anyone can communicate secretly with you ***if they have your certificate***

Modular Arithmetic

- Integers $Z_n = \{0, 1, 2, ..., n-1\}$
- x mod n = remainder of x divided by n
  - 5 mod 13 = 5
  - 13 mod 5 = 3
- y is **modular inverse** of x iff xy mod n = 1
  - 4 is inverse of 3 in Z_{11}
- If n is prime, then Z_n has modular inverses for all integers except 0

Euler's Totient Function

- **coprime**: having no common positive factors other than 1 (also called **relatively prime**)
  - 16 and 25 are coprime
  - 6 and 27 are not coprime
- **Euler's Totient Function**: $\Phi(n)$ = number of integers less than or equal to n that are coprime with n

$$\Phi(n) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$$

where product ranges over distinct primes dividing n

- If m and n are coprime, then $\Phi(mn) = \Phi(m)\Phi(n)$
- If m is prime, then $\Phi(m) = m - 1$

Euler's Totient Function

$$\Phi(n) = n \cdot \prod_{p|n}(1 - \frac{1}{p})$$

$$\Phi(18) = \Phi(3^2 \cdot 2^1) = 18\big(1 - \frac{1}{3}\big)\big(1 - \frac{1}{2}\big) = 6$$

RSA
(Rivest, Shamir, Adelman)

- The dominant public key algorithm
  - The algorithm itself is conceptually simple
  - Why it is secure is very deep (number theory)
  - Uses properties of exponentiation modulo a product of large primes

"A method for obtaining Digital Signatures and Public Key Cryptosystems“, Communications of the ACM, Feb. 1978.

RSA Key Generation

- Choose distinct primes p and q
- Compute $n = pq$
- Compute $\Phi(n) = \Phi(pq) = (p-1)(q-1)$ **WHY?**
- Randomly choose $1<e< \Phi(pq)$ such that e and $\Phi(pq)$ are coprime. e is the **public key exponent**
- Compute $d=e^{-1} \mod(\Phi(pq))$. d is the **private key exponent**

Example:

let $p=3$, $q=11$

$n=33$

$\Phi(pq)=(3-1)(11-1)=20$

let $e=7$

$ed \mod \Phi(pq) = 1$

$7d \mod 20 = 1$

$d = 3$

RSA Encryption/Decryption

- Public key k^+ is $\{e,n\}$ and private key k^- is $\{d,n\}$
- Encryption and Decryption

$$E_{k+}(M) : \text{ciphertext} = \text{plaintext}^e \bmod n$$

$$D_{k-}(\text{ciphertext}) : \text{plaintext} = \text{ciphertext}^d \bmod n$$

- Example
  - Public key (7,33), Private Key (3,33)
  - Plaintext: 4

  - $E(\{7,33\},4) = 4^7 \bmod 33 = 16384 \bmod 33 = 16$
  - $D(\{3,33\},16) = 16^3 \bmod 33 = 4096 \bmod 33 = 4$

Is RSA Secure?

- $\{e, n\}$ is public information
- If you could factor n into $p*q$, then
  - could compute $\phi(n) = (p-1)(q-1)$
  - could compute $d = e^{-1} \bmod \phi(n)$
  - would know the private key $<d,n>$!
- But: factoring large integers is hard!
  - classical problem worked on for centuries; no known reliable, fast method

Why does it work?

- Difficult to find $\Phi(n)$ or d using only e and n.
- Finding d is equivalent in difficulty to factoring n as p*q
  - No efficient integer factorization algorithm is known
  - Example: Took 18 months to factor a 200 digit number into its 2 prime factors
- It is feasible to encrypt and decrypt because:
  - It is possible to find large primes.
  - It is possible to find coprimes and their inverses.
  - Modular exponentiation is feasible.

Security (Cont' d)

- At present, key sizes of 2048 bits are considered to be secure, but 4096 bits is better
- Tips for making n difficult to factor
  1. p and q lengths should be similar (ex.: ~1000 bits each if key is 2048 bits)
  2. both (p-1) and (q-1) should contain a "large" prime factor
  3. gcd(p-1, q-1) should be "small"
  4. d should be larger than $n^{1/4}$

Attacks Against RSA

- Brute force: try all possible private keys
  - can be defeated by using a large enough key space (e.g., 2048 bit keys or larger)
- Mathematical attacks

1. factor *n* (possible for special cases of n)
2. determine *d* directly from e, without computing $\phi(n)$
   - at least as difficult as factoring *n*

Attacks (Cont'd)

- Probable-message attack (using {e,n})
  - encrypt all possible plaintext messages
  - try to find a match between the ciphertext and one of the encrypted messages
  - only works for small plaintext message sizes
- Solution: pad plaintext message with random text before encryption
- PKCS #1 v1 specifies this padding format:

00	02	R1	R2	R3	R4	R5	R6	R7	R8	00	data...

each 8 bits long

Timing Attacks Against RSA

- Recovers the private key from the running time of the decryption algorithm
- Computing $m = c^d \bmod n$ using repeated squaring algorithm:

```
• m = 1;
• for i = k-1 downto 1
    m = m*m mod n;
    if d_i == 1
        then m = m*c mod n;
• return m;
```

Power Analysis Against RSA

- Measure power consumption of the smart card *while it is doing decryption*
- Look at the power spectrum to identify points where more power was used.

Countermeasures to Timing Attacks

1. Delay the result if the computation is too fast
    - disadvantage: ?
2. Add a random delay
    - disadvantage?
3. *Blinding*: multiply the ciphertext by a random number before performing decryption

RSA's Blinding Algorithm

- To confound timing attacks during decryption
  1. generate a random number r between 0 and $n-1$ such that $\gcd(r, n) = 1$ (i.e., co-primes)
  2. compute $\boldsymbol{c}' = \boldsymbol{c} * r^{e} \bmod n$
  3. compute $\boldsymbol{m}' = (\boldsymbol{c}')^{d} \bmod n$
  4. compute $\boldsymbol{m} = \boldsymbol{m}' * r^{-1} \bmod n$

this is where timing attack would occur

- Attacker will not know what the bits of **c'** are
- Performance penalty: < 10% slowdown in decryption speed