RSA (Rivest, Shamir, Adelman)

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

RSA Key Generation

• Pick two large primes $p$ and $q$

• Calculate $n = pq$

• Pick $e$ such that it is relatively prime to $\phi(n) = (q-1)(p-1)$
  – “Euler’s Totient Function”

• $d \approx e^{-1} \mod \phi(n)$
  
  or

  $de \mod \phi(n) = 1$

1. $p=3$, $q=11$

2. $n = 3 \times 11 = 33$

3. $\phi(n) = (2 \times 10) = 20$

4. $e = 7 \mid \gcd(20, 7) = 1$
  “Euclid’s Algorithm”

5. $d = 7-1 \mod 20$
   $d = 7 \mod 20 = 1$
   $d = 3$
RSA Encryption/Decryption

- Public key $k^+$ is $\{e,n\}$ and private key $k^-$ is $\{d,n\}$
- Encryption and Decryption
  \[
  E(k^+, P) : \text{ciphertext} = \text{plaintext}^e \mod n
  \]
  \[
  D(k^-, C) : \text{plaintext} = \text{ciphertext}^d \mod n
  \]
- Example
  - Public key (7,33), Private Key (3,33)
  - Data “4” (encoding of actual data)
    - $E(\{7,33\}, 4) = 4^7 \mod 33 = 16384 \mod 33 = 16$
    - $D(\{3,33\}, 16) = 16^3 \mod 33 = 4096 \mod 33 = 4$
Encryption using private key ...

• Encryption and Decryption
  \[ E(k^-, P) : \text{ciphertext} = \text{plaintext}^d \mod n \]
  \[ D(k^+, C) : \text{plaintext} = \text{ciphertext}^e \mod n \]

• E.g.,
  
  \[ E(\{3, 45\}, 4) = 4^3 \mod 33 = 64 \mod 33 = 31 \]
  
  \[ D(\{7, 45\}, 19) = 31^7 \mod 33 = 27,512,614,111 \mod 33 = 4 \]

• Q: Why encrypt with private key?
Applied Cryptography

- How do we use these (and other) constructs to achieve security goals?
  - What are the goals?
Meet Alice and Bob ….

• *Alice* and *Bob* are the canonical players in the cryptographic world.
  – They represent the end points of some interaction
  – Used to illustrate/define a security protocol

• Other players occasionally join …
  – Trent - trusted third party
  – Mallory - malicious entity
  – Eve - eavesdropper
  – Ivan - an issuer (of some object)
Some notation …

- You will generally see protocols defined in terms of exchanges containing some notation like
  - All players are identified by their first initial
    - E.g., Alice=A, Bob=B
  - $d$ is some data
  - $pw^A$ is the password for A
  - $k^{AB}$ is a symmetric key known to A and B
  - $A^+,A^-$ is a public/private key pair for entity A
  - $E(k,d)$ is encryption of data $d$ with key $k$
  - $H(d)$ is the hash of data $d$
  - $\text{Sig}(A^-,d)$ is the signature (using A’s private key) of data $d$
  - “+” is used to refer to concatenation
Some interesting things you want to do …

• … when communicating.
  – Ensure the **authenticity** of a user
  – Ensure the **integrity** of the data
    • Also called **data authenticity**
  – Keep data **confidential**
  – Guarantee **non-repudiation**
Basic (User) Authentication

- Bob wants to authenticate Alice’s identity
  - (is who she says she is)

\[ \text{pw}^A \]

\[ \text{Y/N} \]
Hash User Authentication

- Bob wants to authenticate Alice’s identity
  - (is who she says she is)

\[ h(pw^A) \]

1. Alice
2. Bob

[Y/N]
Challenge/Response User Authentication

- Bob wants to authenticate Alice’s identity
  - (is who she says she is)

\[ h(c + pw^A) \]

\[ Y/N \]
User Authentication vs. Data Integrity

• User authentication proves a property about the communicating parties
  – E.g., I know a password

• Data integrity ensures that the data transmitted...
  – Can be verified to be from an authenticated user
  – Can be verified to determine whether it has been modified

• Now, let's talk about the latter, data integrity
Simple Data Integrity?

- Alice wants to ensure any modification of the data in flight is detectable by Bob (integrity)
HMAC Integrity

- Alice wants to ensure any modification of the data in flight is detectable by Bob (integrity)

\[ [d, h(d + pw^A)] \]
Signature Integrity

- Alice wants to ensure any modification of the data in flight is detectable by Bob (integrity)

\[ [d, \text{Sig}(A,d)] \]
Data Integrity vs. Non-repudiation

• If the integrity of the data is preserved, is it provably from that source?
  – Hash integrity says what about non-repudiation?
  – Signature integrity says what about non-repudiation?
Confidentiality

- Alice wants to ensure that the data is not exposed to anyone except the intended recipient (confidentiality)
Confidentiality

• Alice wants to ensure that the data is not exposed to anyone except the intended recipient (confidentiality)
• But, Alice and Bob have *never met*!!!

Alice randomly selects key $k^x$ to encrypt with

\[ [E(k^x,d), E(B^+, k^x)] \]

• Alice randomly selects key $k^x$ to encrypt with
Real Systems Security

• The reality of the security is that 90% of the frequently used protocols use some variant of these constructs.
  – So, get to know them … they are your friends
  – We will see them (and a few more) over the semester

• They also apply to systems construction
  – Protocols need not necessarily be online
  – Think about how you would use these constructs to secure files on a disk drive (integrity, authenticity, confidentiality)
  – We will add some other tools, but these are the basics
Using hash values as authenticators

- Consider the following scenario
  - Alice is a teacher who 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
  - Alice does not trust Mallory to deliver the message.
- She and Bob use the following protocol:
  1. Alice invents a secret $t$
  2. Alice gives Bob $h(t)$, where $h()$ is a crypto hash function
  3. 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 Alice 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 $h(t)$ (but we assume Bob is smart enough to come to that conclusion when the room is empty)

• What is important here is that hash preimages are good as (single bit) authenticators.

• Note that it is important that Bob got 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 applications 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 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 produce $t$ 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, because they are all post-images of today’s value
  
    – Note: Mallory can again convince Bob that class is occurring by not delivering $H^{(26-d)}(t)$
  
• Important: chain of hash values are ordered authenticators

• Important that Bob got the original value $H^{26}(t)$ from Alice directly (was provably authentic)
Proof Systems

• Consider the following scenario
  • Alice wants to tell Bob the identity of the students in her class
  • Alice again must use Mallory as a go between
  • Mallory wants to add students to the class (presumably because she gets paid by the student)
  • Assume Bob has Alice’s public key only

• Membership problem: prove set \( S=\{a_1, a_2, \ldots, a_n\} \), where \( a_x \) is an element (in this case a student)
  • Alice acts as a *prover*, where she proves particular \( a_x \) objects are in \( S \)
  • Bob acts as a *validator*, where he assesses membership
  • The device Alice uses to prove membership is a *proof system*
Simple Proof Systems

- **List proof**: In the most obvious case, Alice simply signs the whole list and gives it to Bob
  - E.g., $\text{Sig}\{a_1, a_2, \ldots, a_n\}, A^-$

- **Element proof**: Alice may also create proofs for each element in $S$
  - E.g., $\text{Sig}\{a_1\}, A^-$, $\text{Sig}\{a_2\}, A^-$, $\ldots$, $\text{Sig}\{a_n\}, A^-$

- Problem:
  - it is costly to transfer lists: they can be BIG and may not change much (but may change some)
  - and too costly to validate Element proofs, you have to validate each signature independently, and you need many

- Is there another way?
Tree Proof System

- Arrange the elements in a tree, hash toward the root. Consider
  - \( S = (\text{Joe, Steve, Mark, Nate}) \)

- Alice signs the root value
  - \( \text{Sign}(H(H(J+S)+H(M+N)), A^-) \)

- Alice gives Mallory a proof for Joe
  - All the sibling values on the path
    - \( \text{Sign}(H(H(J+S)+H(M+N)), A^-), H(M+N), S \)

- Alice gives a similar proof for all other students in S
Tree Proof System

• Why does this work?
  - Mallory cannot find any student name that would match any hash values (if she could, she could find a collision)

• Advantage:
  - You can amortize the cost of validation (because there is only only one signature over all proofs)
  - The size of the proofs is small (it is logarithmic in the size of number of elements of S)

• So, you get the best of both worlds!
Authenticated Dictionaries

- 2-3 tree with Merkle hashing

\[ H(L, M, R) \]

\[ H(\text{proof}) \]

\[ H(\text{proof}_1 | \text{proof}_2) \]
Key Distribution

• Key Distribution is the process where we assign and transfer keys to a participant
  – Out of band (e.g., passwords, simple)
  – During authentication (e.g., Kerberos)
  – As part of communication (e.g., skip-encryption)

• Key agreement is the process whereby two parties negotiate a key
  – 2 or more participants

• Typically, key distribution/agreement this occurs in conjunction with or after authentication.
  – However, many applications can pre-load keys
Diffie-Hellman Key Agreement

• The DH paper really started the modern age of cryptography, and indirectly the security community
  – Negotiate a secret over an insecure media
  – E.g., “in the clear” (seems impossible)
  – Idea: participants exchange intractable puzzles that can be solved easily with additional information.

• Mathematics are very deep
  – Working in multiplicative group G
  – Use the hardness of computing discrete logarithms in finite field to make secure
  – Things like RSA are variants that exploit similar properties
Diffie-Hellman Protocol

• For two participants $p^1$ and $p^2$
• Setup: We pick a prime number $p$ and a base $g (<p)$
  – This information is public
  – E.g., $p=13$, $g=4$
• Step 1: Each principal picks a private value $x (<p-1)$
• Step 2: Each principal generates and communicates a new value
  \[ y = g^x \mod p \]
• Step 3: Each principal generates the secret shared key $z$
  \[ z = y^x \mod p \]
• Perform a neighbor exchange.
Attacks on Diffie-Hellman

• This is key exchange, not authentication.
  – You really don’t know anything about who you have exchanged keys with
  – The man in the middle …

  – Alice and Bob think they are talking directly to each other, but Mallory is actually performing two separate exchanges

• You need to have an authenticated DH exchange
  – The parties sign the exchanges (more or less)
  – See Schneier for a intuitive description
Question

• If I already have an authenticated channel (e.g., the remote party’s public key), why don’t I simply make up a key and send it to them?