Introduction to Cryptography

Lecture 9

Benny Pinkas

Integer Multiplication & Factoring as a One Way Function.

$$\begin{tikzcd}
p,q \arrow[r, bend left, "\textbf{easy}"] & N=pq \arrow[l, bend left, "\textbf{hard}"]
\end{tikzcd}$$

Can a public key system be based on this observation ?????

The Multiplicative Group Z_{pq}*

- *p* and *q* denote two large primes (e.g. 512 bits long).
- Denote their product as $N = pq$.
- The multiplicative group $Z_N^* = Z_{pq}^*$ contains all integers in the range *[1,pq-1]* that are relatively prime to both *p* and *q*.

- The size of the group is
  - $\phi(n) = \phi(pq) = (p-1)(q-1) = N - (p+q) + 1$

- For every $x \in Z_N^*$, $x^{\phi(N)} = x^{(p-1)(q-1)} = 1 \bmod N$.

Trapdoor permutation

- A trapdoor permutation is a tuple of three PPT (Probabilistic Polynomial Time) algorithms:
  - GEN(1^k): Outputs a pair (F,F^{-1})
  - F is a permutation over $\{0,1\}^k$. (In our case the permutation is over Z_n^*.)
  - Correctness: $F^{-1}(F(x)) = x$.
  - One-wayness: For all PPT A, for (F,F^{-1}) randomly generated by GEN, and random x, the probability that $A(F,F(x))=x$ is negligible.

  - In other words, inverting F without the trapdoor F^{-1} is hard.

Example

- $f_{g,p}(x) = g^x \bmod p$ is *not* a trapdoor one way function.
  - Why?
- Therefore El Gamal encryption is not based on assuming the existence of a trapdoor one way function.

The RSA Public Key Cryptosystem Trapdoor Permutation

- The RSA function (textbook RSA) is not a secure encryption system
  - Does not satisfy basic security definitions
  - Many attacks do exist

- It implements a trapdoor permutation, which is the basis for secure public key encryption
  - Is the working horse of public key cryptography

The RSA Public Key Cryptosystem Trapdoor Permutation

- Gen (public key):
  - $N=pq$ the product of two primes (we assume that factoring N is hard)
  - e such that gcd$(e,\phi(N))=1$ *(are these hard to find?)*
- Trapdoor (Private key):
  - d such that $de\equiv 1$ mod $\phi(N)$

- Computing F (Encryption) of $M \in Z_N^*$
  - $C=E(M)=M^e$ mod N
- Computing F^{-1} (Decryption) of $C \in Z_N^*$
  - $M=D(C)=C^d$ mod N *(why does it work?)*

Security reductions

- Security by reduction
  - Define what it means for the system to be “secure” (chosen plaintext/ciphertext attacks, etc.)
  - State a “hardness assumption” (e.g., that it is hard to extract discrete logarithms in a certain group).
  - Show that if the hardness assumption holds then the cryptosystem is secure.
- Benefits:
  - To examine the security of the system it is sufficient to check whether the assumption holds
  - Similarly, for setting parameters (e.g. group size).

RSA Security

- (For ElGamal encryption, we showed that if the DDH assumption holds then El Gamal encryption has semantic security.)
- We know that if factoring N is easy then RSA is insecure
  - can factor $N \Rightarrow$ find $p,q \Rightarrow$ find $(p-1)(q-1) \Rightarrow$ find d from $e \Rightarrow$ decrypt RSA
  - Is the converse true? (we would have liked to show that decrypting RSA $\Rightarrow$ factoring N)
- Factoring assumption:
  - For a randomly chosen p,q of good length, it is infeasible to factor $N=pq$.
  - This assumption might be too weak (might not ensure secure RSA encryption)
  - Maybe it is possible to break RSA without factoring N ?
  - We don’t know how to reduce RSA security to the hardness of factoring.

  - Fact: finding d is equivalent to factoring (will not be proved here)
  - I.e., if it is possible to find d given (N,e) , then it is easy to factor $N.$
  - can find d from $e \;\Rightarrow\;$ can factor N
  - But perhaps it is possible to break RSA without finding d?

The RSA assumption: Trap-Door One-Way Function (OWF)

- (what is the minimal assumption required to show that RSA encryption is secure?)

The RSA assumption: Trap-Door One-Way Function (OWF)

- The RSA assumption: the RSA function is a trapdoor permutation
  - The setting: Generate random RSA keys *(N,e,d).* Choose random $y \in Z^*_N$. Provide the adversary with *N,e,y.*
  - The assumption that is the there is no efficient algorithm which can output x such that $x^e = y$ mod *N.*
  - (The trapdoor is *d* s.t. $ed = 1$ mod $\phi(N)$)

- More concretely, (n,e,t,ε)-RSA assumption
  - For all t-time A
  - Choose p,q as random n/2 bit primes, N=pq, choose random x in Z_N^*.
  - Pr (A(N,e,y) = x, s.t. x^e=y mod N) < ε

RSA as a One Way Trapdoor Permutation

$$\begin{tikzcd}
x \arrow[r, bend left, "\text{easy}"] & x^e \bmod N \arrow[l, bend left, "\text{hard}"] \arrow[l, bend left=80, "\text{Easy with trapdoor info (d)}"]
\end{tikzcd}$$

RSA assumption: cautions

- The RSA assumption is quite well established:
  - Namely, the assumption that RSA is a Trapdoor One-Way Permutation
  - This means that it is hard to invert RSA on a random input – without knowing the secret key

- But is it a secure cryptosystem?
  - Given the assumption it is hard to reconstruct the input x (if x was chosen randomly), but is it hard to learn *anything* about x?

- Theorem [G]: RSA hides the log(log(*n*)) least *and* most significant bits of a uniformly-distributed random input
  - But some (other) information about pre-image may leak

Security of RSA

- Deterministic encryption. In textbook RSA:
  - M is always encrypted as M^e
  - The ciphertext is as long as the domain of M
- Corollary: textbook RSA does not have semantic security.
  - If we suspect that a ciphertext is an encryption of a specific message m, we can encrypt m and compare it to the ciphertext. If the result is equal, then m is indeed the message encrypted in the ciphertext.
- In the last recitation we showed that if M is a 64 bit message, it is easy tor recover it using a meet in the middle attack.

- Encrypting random messages:
- It can be proved that if the message M is chosen uniformly at random from Z^*_N, then the RSA assumption means that no efficient algorithm can recover M from N,e,M^e.

Security of RSA

- Chosen ciphertext attack: (homomorphic property)
  - Given $C = M^e$ and $C' = M'^{\,e}$ it is easy to compute C''=$MM'^{\,e}$
  - Textbook RSA is therefore also susceptible to chosen ciphertext attacks:
    - We are given a ciphertext $C=M^e$
    - We can choose a random R and generate $C'=CR^e$ (an encryption of $M\cdot R$).
    - Suppose we can receive the decryption of C'. It is equal to $M\cdot R$.
    - We divide it by R and reveal M.

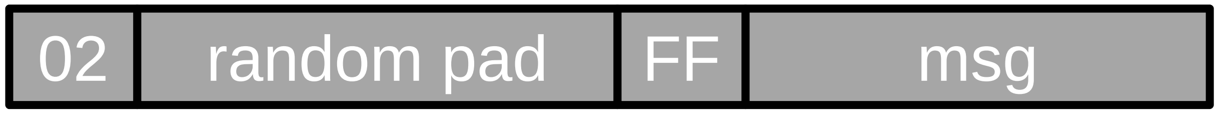
Padded RSA

- In order to make textbook RSA semantically secure we must change it to be a probabilistic encryption
  - The initial message must be preprocessed before being input to the RSA function
  - For example, we can pad the message with random bits.
    - Suppose that messages are of length $|N|-L$ bits
    - To encrypt a message M, choose a random string r of length L, and compute $(r \mid M)^e$ mod N.
    - When decrypting, output only the last $|N|-L$ bits of C^d mod N

- Any message has 2^L possible encryptions. L must be long enough so that a search of all 2^L pads is inefficient.
- There is no known proof that this secure.
- Similar schemes can be proven to be secure under certain assumptions

RSA in practice – PKCS1 V1.5

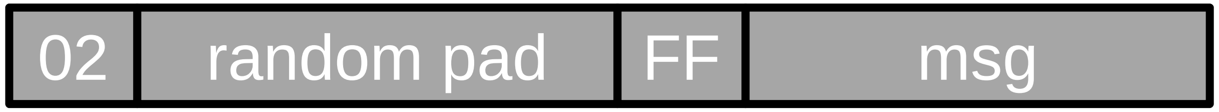
- To encrypt a message



  - The resulted is encrypted using the RSA function
  - This system is widely deployed even though it has no security analysis

PKCS1 V1.5 – Attack [Bleinchenbacher 98]

- To encrypt a message



- PKCS1 as used in SSL
  - Server decrypts message. If first byte is not 02, sends an error message.
  - Attacker can test if plaintext begins with “02”
- Attack:
  - Given ciphertext C, choose random r. Compute $C' = r^e C = (r \cdot \text{PKCS1}(msg))^e$.
  - Send C’ and wait for response.

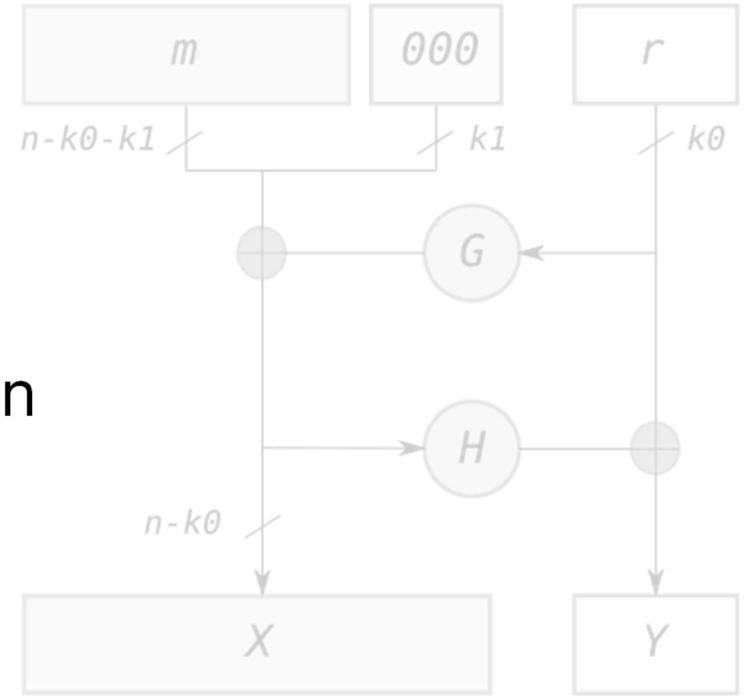
PKCS1 V2.0 – OAEP (based on slides by Dan Boneh)

- OAEP (Optimal asymmetric encryption padding)

- Encrypt X|Y using RSA
- Decryption: check pad and reject if invalid.

Thm: If RSA is a trapdoor permutation then RSA-OAEP provides chosen ciphertext security when H,G are "random oracles".

Usually implement H,G using SHA.



Implementation attacks (based on slides by Dan Boneh)

- Attack the implementation of RSA
- Timing attack (Kocher 97)
  - The time it takes to compute C^d mod N can expose d.
- Power attack (Kocher 99)
  - The power consumption of a smartcard while it is computing C^d mod N can expose d.
- Faults attack: (BDL 97) (presented last week)
  - A computer error during C^d mod N can expose d.
  - OpenSSL defense: check output. 5% slowdown.

Digital Signatures

Handwritten signatures

- Associate a document with an signer (individual)
- Signature can be verified against a different signature of the individual
- It is hard to forge the signature…
- It is hard to change the document after it was signed…
- Signatures are legally binding

Desiderata for digital signatures

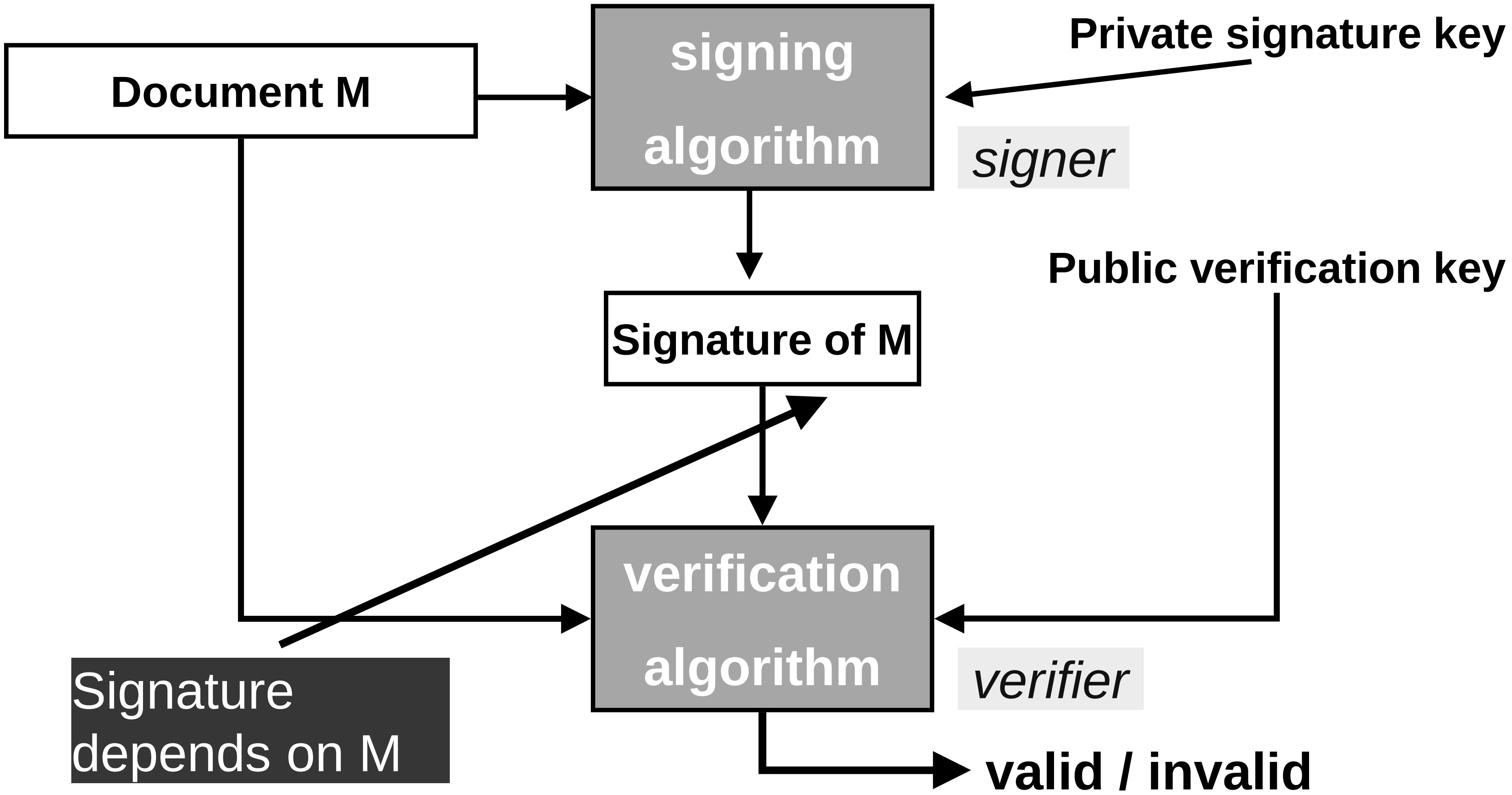
- Associate a document to an signer
- A digital signature is attached to a document *(rather then be part of it)*
- The signature is easy to verify but hard to forge
  - Signing is done using knowledge of a private key
  - Verification is done using a public key associated with the signer *(rather than comparing to an original signature)*
  - It is impossible to change even one bit in the signed document
- *A copy of a digitally signed document is as good as the original signed document.*
- Digital signatures could be legally binding…

Non Repudiation

- Prevent signer from denying that it signed the message
- I.e., the receiver can prove to third parties that the message was signed by the signer

- This is different than message authentication (MACs)
  - There the receiver is assured that the message was sent by the receiver and was not changed in transit
  - But the receiver cannot prove this to other parties
    - MACs: sender and receiver share a secret key *K*
    - If R sees a message MACed with *K*, it knows that it could have only been generated by S
    - But if R shows the MAC to a third party, it cannot prove that the MAC was generated by S and not by R

Signing/verification process



Diffie-Hellman
“New directions in cryptography” (1976)

- In public key encryption
  - The encryption function is a trapdoor permutation *f*
    - Everyone can encrypt = compute *f().* (using the public key)
    - Only Alice can decrypt = compute $f^{-1}()$. (using her private key)
- Alice can use *f* for signing
  - Alice signs m by computing $s=f^{-1}(m)$.
  - Verification is done by computing $m=f(s)$.

- Intuition: since only Alice can compute $f^{-1}()$, forgery is infeasible.
- Caveat: none of the established practical signature schemes following this paradigm is provably secure

Example: simple RSA based signatures

- Key generation: (as in RSA)
  - Alice picks random p,q. Finds $e \cdot d=1$ mod $(p-1)(q-1)$.
  - Public verification key: (N,e)
  - Private signature key: d

- Signing: Given m, Alice computes $s=m^d$ mod N.

- Verification: given m,s and public key (N,e).
  - Compute $m' = s^e$ mod N.
  - Output “valid” iff m’=m.

Example: simple RSA based signatures

- Key generation: (as in RSA)
  - Alice picks random *p,q*. Finds $e \cdot d=1$ mod $(p-1)(q-1)$.
  - Public verification key: *(N,e)*
  - Private signature key: *d*

- Signing: Given *m*, Alice computes $s=m^d$ mod *N.*

- Verification: given *m,s* and public key *(N,e).*
  - Compute $m' = s^e$ mod *N*.
  - Output “valid” iff m’=m.

Message lengths

- A technical problem:
  - |m| might be longer than |N|
  - *m* might not be in the domain of $f^{-1}()$

Solution "hash-and-sign" paradigm:

- Signing: First compute *H(m)*, then compute the signature $f^{-1}(H(M))$. Where,
  - The range of *H()* must be contained in the domain of $f^{-1}()$.
  - *H()* must be collision intractable. I.e. it is hard to find (in polynomial time) messages *m, m'* s.t. *H(m)=H(m')*.
- Verification:
  - Compute *f(s).* Compare to *H(m).*

- Using *H()* is also good for security reasons. See below.

Security of using a hash function

- Intuitively
  - Adversary can compute $H(), f(),$ but not $H^{-1}(), f^{-1}()$.
  - Can only compute $(m,H(m))$ by choosing m and computing $H()$.
  - Adversary wants to compute $(m, f^{-1}(H(m)))$.
  - To break signature needs to show s s.t. $f(s)=H(m)$. (E.g. $s^e=H(m)$.)

  - Failed attack strategy 1:
    - Pick s, compute $f(s)$, and look for m s.t. $H(m)=f(s)$.
  - Failed attack strategy 2:
    - Pick m,m' s.t. $H(m)=H(m')$. Ask for a signature s of m' (which is also a signature of m).
    - (If $H()$ is not collision resistant, adversary could find m,m' s.t. $H(m) = H(m')$.)
  - This does not mean that the scheme is secure, only that these attacks fail.