

Introduction to Cryptography

Lecture 11

Factoring, computing discrete logs
SSL / TLS

Benny Pinkas


Integer factorization

- The RSA and Rabin cryptosystems use a modulus N and are insecure if it is possible to factor N .
- Factorization: given N find all prime factors of N .
- Factoring is the search problem corresponding to the primality testing decision problem.
 - Primality testing is easy
 - What about factoring?

Pollard's Rho method

- Factoring N
- Trivial algorithm: trial division by all integers $< N^{1/2}$.
- Pollard's rho method:
 - $O(N^{1/4})$ computation.
 - $O(1)$ memory.
 - A heuristic algorithm.

Pollard's rho method

1. $i=1; x_1 \in [1, n-1]; y=x_1;$
2. $i = i+1.$
3. $x_i = ((x_{i-1})^2 - 1) \bmod n.$
4. $d = \gcd(y-x_i, n)$
5. If $d>1$ then output d , and stop.  Always a factor of n
6. If i is a power of 2, then $y=x_i$
7. Goto line 2.

- x_i is a series of numbers in $0..n-1$.
- y takes the values of $x_1, x_2, x_4, x_8, \dots, x_{2^j}, \dots$
- If $(y-x_i) = 0 \bmod p$, then most likely $\gcd(y-x_i, n)=p$.

Pollard's rho method

- The running time is not guaranteed, but is expected to be $\text{sqrt}(p) \leq n^{1/4}$.
- The sequence x_i is in $1..n$.
 - x_i depends only on x_{i-1} ($x_i = ((x_{i-1})^2 - 1) \bmod n$)
 - The sequence is shaped like the letter Rho.
 - Assume that $f_n(x) = x^2 - 1 \bmod n$ behaves like a random function. Then the tail and the circle are about $\text{sqrt}(n)$ long.
- Let $x'_i = x_i \bmod p$, where p factors n .
- $x'_{i+1} = x_{i+1} \bmod p = (x_i^2 - 1 \bmod n) \bmod p = x_i^2 - 1 \bmod p = (x'_i)^2 - 1 \bmod p$
- The sequence x'_i therefore follows x_i , but is in $0..p-1$. Therefore, its tail and circle are about $\text{sqrt}(p)$ long.

Pollard's rho method

- The sequence x'_j :
 - Let t be the first repeated value in x'_j
 - Let u be the length of the cycle
 - $x'_{t+i} = x'_{t+i+u}$
 - Therefore $x_{t+i} = x_{t+i+u} \bmod p$
 - $\gcd(x_{t+i} - x_{t+i+u}, n) = cp$.
- Once the algorithm saves $y=x_j$ for $j>t$, it is on the circle. If the circle length u is smaller than j , the algorithm computes $\gcd(x_{j+u}-x_j, n)$ and factors n .
- The algorithm fails if
 - The cycle and tail are long \rightarrow running time is slow.
 - The cycle and tail are of the same length for both p and q .

Modern factoring algorithms

- The number-theoretic running time function $L_n(a, c)$

$$L_n(a, c) = e^{c(\ln n)^a (\ln \ln n)^{1-a}}$$

- For $a=0$, the running time is polynomial in $\ln(n)$.
 - For $a=1$, the running time is exponential in $\ln(n)$.
 - For $0 < a < 1$, the running time is subexponential.
- Factoring algorithms
 - Quadratic field sieve: $L_n(1/2, 1)$
 - General number field sieve: $L_n(1/3, 1.9323)$
 - Elliptic curve method $L_p(1/2, 1.41)$ (preferable only if $p \ll \sqrt{n}$)

Modulus size recommendations

- Factoring algorithms are run on massively distributed networks of computers (running in their idle time).
- RSA published a list of factoring challenges.
- A 512 bit challenge was factored in 1999.
- The largest factored number $n=pq$.
 - 640 bits (RSA-640)
 - Factored on November 2, 2005 using the NFS
- Typical current choices:
 - At least 768-bit RSA moduli should be used
 - For better security, 1024-bit RSA moduli are used
 - For more sensitive applications, key lengths of 2048 bits (or higher) are used

RSA with a modulus with more factors

- The best factoring algorithms:
 - General number field sieve (NFS): $L_n(1/3, 1.9323)$
 - Elliptic curve method $L_p(1/2, 1.41)$
- If $n=pq$, where $|p|=|q|$, then the NFS is faster.
 - Common parameters: $|p|=|q|=512$ bits
 - Factoring using the NFS is infeasible, but more likely than factoring using the elliptic curve method.
- How about using $N=pqr$, where $|p|=|q|=|r|=512$?
 - The factors are of the same length, so factoring using the elliptic curve method is still infeasible.
 - The NFS method has to work on a larger modulus
 - Decryption time is slower.

RSA for paranoids

- Suppose $N=pq$, $|p|=500$ bits, $|q|=4500$ bits.
- Factoring is extremely hard.
- Decryption is also very slow. (Encryption is done using a short exponent, so it is pretty efficient.)
- However, in most applications RSA is used to transfer session keys, which are rather short.
- Assume message length is < 500 bits.
 - In the decryption process, it is only required to decrypt the message modulo p . (As, or more, efficient, as a 1024 bit n .)
 - Encryption must use a slightly longer e . Say, $e=20$.

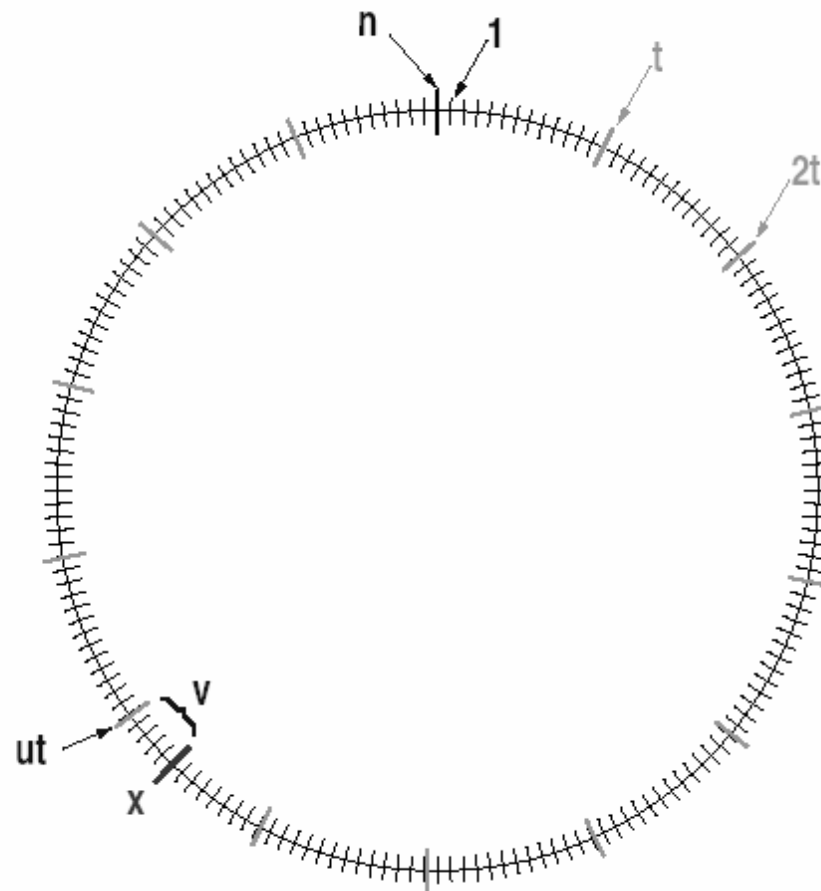
Discrete log algorithms

- Input: (g, y) in a finite group G . Output: x s.t. $g^x = y$ in G .
- Generic vs. special purpose algorithms: generic algorithms do not exploit the representation of group elements.
- Algorithms
 - Baby-step giant-step: Generic. $|G|$ can be unknown. $\text{Sqrt}(|G|)$ running time and memory.
 - Pollard's rho method: Generic. $|G|$ must be known. $\text{Sqrt}(|G|)$ running time and $O(1)$ memory.
 - No generic algorithm can do better than $O(\text{sqrt}(q))$, where q is the largest prime factor of $|G|$
 - Pohlig-Hellman: Generic. $|G|$ and its factorization must be known. $O(\text{sqrt}(q) \ln q)$, where q is largest prime factor of $|G|$.
 - Therefore for \mathbb{Z}_p^* , $p-1$ must have a large prime factor.
 - Index calculus algorithm for \mathbb{Z}_p^* : $L(1/2, c)$
 - Number field sieve for \mathbb{Z}_p^* : $L(1/3, 1.923)$

Baby-step giant-step DL algorithm

- Let $t = \sqrt{|G|}$.
- x can be represented as $x = ut - v$, where $u, v < \sqrt{|G|}$.
- The algorithm:
 - Giant step: compute the pairs (j, g^{jt}) , for $0 \leq j \leq t$. Store in a table keyed by g^{jt} .
 - Baby step: compute $y \cdot g^i$ for $i = 0, 1, 2, \dots$, until you hit an item (j, g^{jt}) in the table. $x = jt - i$.
- Memory and running time are $O(\sqrt{|G|})$.

Baby-step giant-step DL algorithm



SSL/TLS

- General structure of secure HTTP connections
 - To connect to a secure web site using SSL or TLS, we send an https:// command
 - The web site sends back a public key⁽¹⁾, and a certificate.
 - Our browser
 - Checks that the certificate belongs to the url we're visiting
 - Checks the expiration date
 - Checks that the certificate is signed by a CA whose public key is known to the browser
 - Checks the signature
 - If everything is fine, it chooses a session key and sends it to the server encrypted with RSA using the server's public key

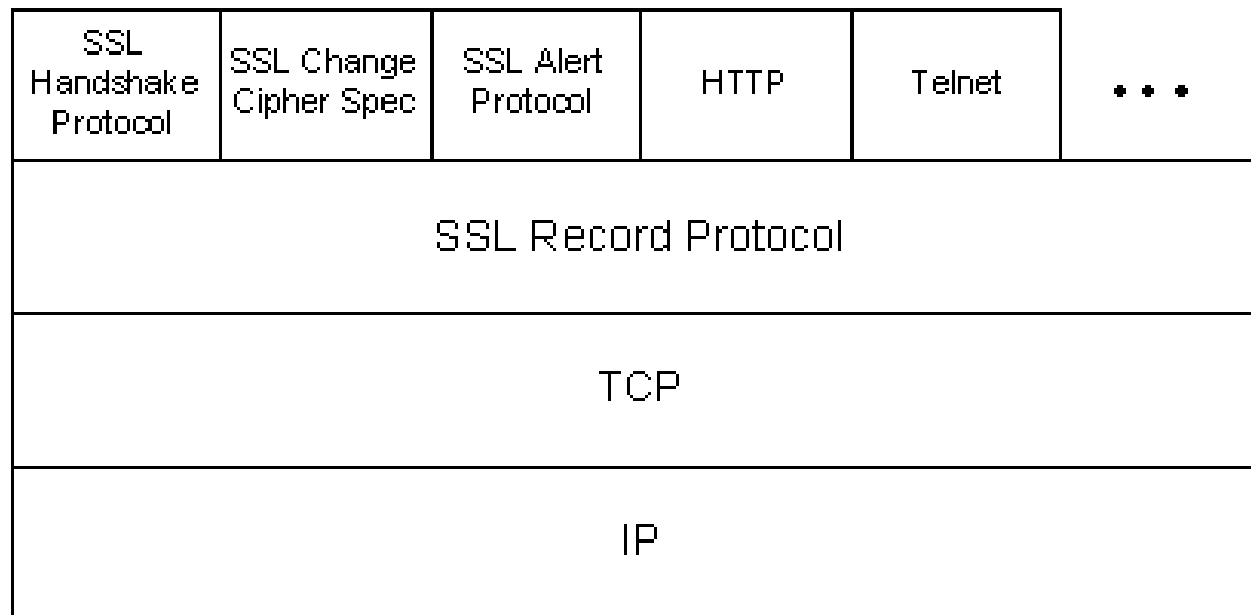
⁽¹⁾ This is a very simplified version of the actual protocol.

SSL/TLS

- SSL (Secure Sockets Layer)
 - SSL v2
 - Released in 1995 with Netscape 1.1
 - A flaw found in the key generation algorithm
 - SSL v3
 - Improved, released in 1996
 - Public design process
- TLS (Transport Layer Security)
 - IETF standard, RFC 2246
- Common browsers support all these protocols

SSL Protocol Stack

- SSL/TLS operates over TCP, which ensures reliable transport.
- Supports any application protocol (usually used with http).



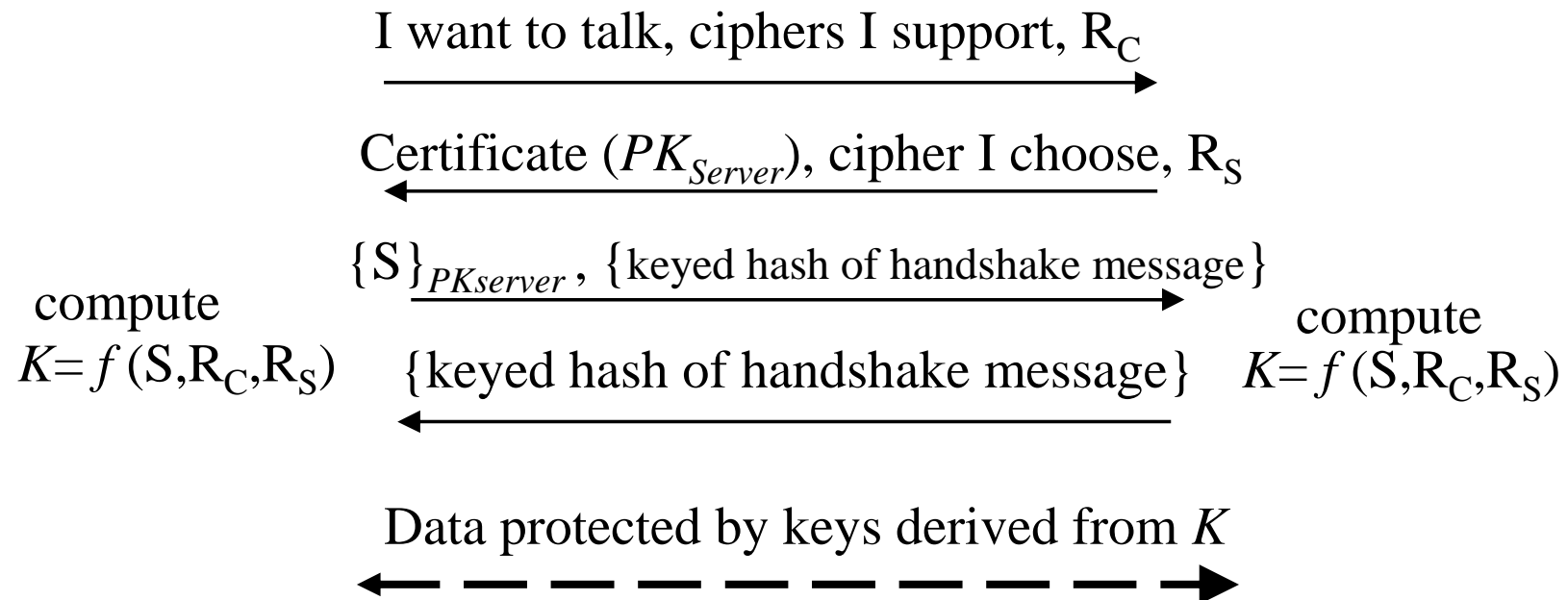
SSL/TLS Overview

- Handshake Protocol - establishes a session
 - Agreement on algorithms and security parameters
 - Identity authentication
 - Agreement on a key
 - Report error conditions to each other
- Record Protocol - Secures the transferred data
 - Message encryption and authentication
- Alert Protocol – Error notification (including “fatal” errors).
- Change Cipher Protocol – Activates the pending crypto suite

Simplified SSL Handshake

Client

Server



A typical run of a TLS protocol

- $C \Rightarrow S$
 - ClientHello.protocol.version = “TLS version 1.0”
 - ClientHello.random = T_C, N_C
 - ClientHello.session_id = “NULL”
 - ClientHello.crypto_suite = “RSA: encryption.SHA-1:HMAC”
 - ClientHello.compression_method = “NULL”
- $S \Rightarrow C$
 - ServerHello.protocol.version = “TLS version 1.0”
 - ServerHello.random = T_S, N_S
 - ServerHello.session_id = “1234”
 - ServerHello.crypto_suite = “RSA: encryption.SHA-1:HMAC”
 - ServerHello.compression_method = “NULL”
 - ServerCertificate = pointer to server’s certificate

Some additional issues

- More on $S \Rightarrow C$
 - The ServerHello message can also contain Certificate Request Message
 - I.e., server may request client to send its certificate
 - Two fields: certificate type and acceptable CAs
- Negotiating crypto suites
 - The crypto suite defines the encryption and authentication algorithms and the key lengths to be used.
 - ~30 predefined standard crypto suites
 - Selection (SSL v3): Client proposes a set of suites. Server selects one.

Key generation

- Key computation:
 - The key is generated in two steps:
 - *pre-master secret* S is exchanged during handshake
 - *master secret* K is a 48 byte value calculated using pre-master secret and the random nonces
- Session vs. Connection: a *session* is relatively long lived. Multiple TCP *connections* can be supported under the same SSL/TSL connection.
- For each connection: 6 keys are generated from the master secret K and from the nonces. (For each direction: encryption key, authentication key, IV.)

TLS Record Protocol

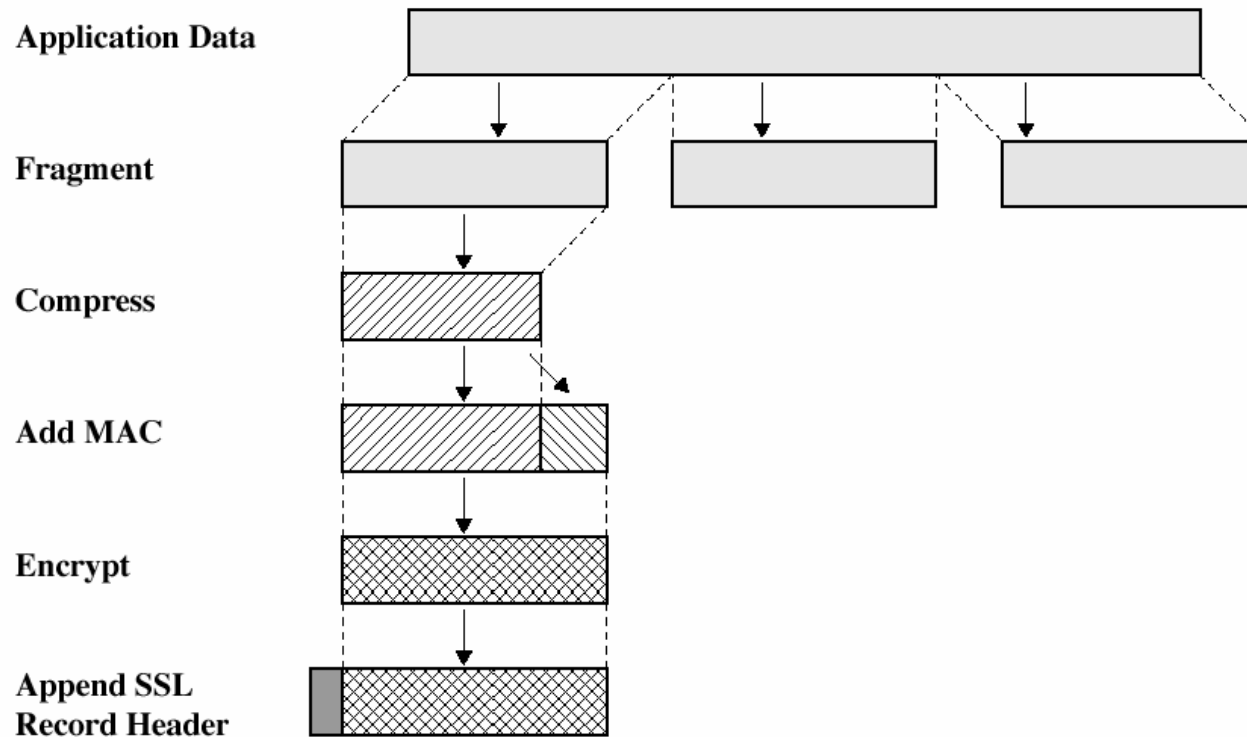


Figure 17.3 SSL Record Protocol Operation