CHAPTER 13 – FROM CRYPTO-THEORY TO CRYPTO-PRACTICE II

In this chapter we deal in more details with several new practical issues of contemporary cryptography as well as with several new problems:

Namely, we deal with following topics:

- RSA – from theory to practice
- Stream cryptosystems
- Electronic voting
- Digital cash
- Anonymity protocols
- Privacy preservation
- Key agreement on networks

VARIATIONS on RSA

RSA cryptosystem is the most important public-key cryptosystems and therefore it has been analyzed carefully. In the following we discuss the following related problems:

- Randomized version of RSA that is semantically secure (what does not hold for standard version of RSA.

- Cases when one can break RSA

- RSA standard

- Special attacks on RSA

To start with we repeat basic description of RSA.

DESIGN and USE of RSA CRYPTOSYSTEM

Invented in 1978 by Rivest, Shamir, Adleman

Basic idea: prime multiplication is very easy, integer factorization seems to be unfeasible.

Design of RSA cryptosystems

1. Choose two large s-bit primes *p,q, s in [512,1024],* and denote

$$n = pq,\ \phi(n) = (p-1)(q-1)$$

2. Choose a large *d* such that

$$\gcd(d, \phi(n)) = 1$$

and compute

$$e = d^{-1} (\bmod \phi(n))$$

Public key: *n* (modulus), *e* (encryption algorithm)

Trapdoor information: *p, q, d* (decryption algorithm)

Plaintext *w*

Encryption: cryptotext $c = w^e \bmod n$

Decryption: plaintext $w = c^d \bmod n$

Details: A plaintext is first encoded as a word over the alphabet {0, 1,…,9}, then divided into blocks of length *i* -1, where $10^{i-1} < n < 10^{i}$. Each block is taken as an integer and decrypted using modular exponentiation.

Randomized version of RSA-like cryptosystems

The scheme works for any trapdoor function (as in case of RSA),

$$f : D \to D, D \subset \{0,1\}^n,$$

for any pseudorandom generator

$$G: \{0,1\}^k \to \{0,1\}^l,\ k << l$$

and any hash function

$$h: \{0,1\}^l \to \{0,1\}^k,$$

where $n = l + k$. Given a random seed $s \in \{0,1\}^k$ as input, G generates a pseudorandom bit-sequence of length l.

Encryption of a message $m \in \{0,1\}^l$ is done as follows:

1. A random string $r \in \{0,1\}^k$ is chosen.
2. Set $x = (m \oplus G(r)) \| (r \oplus h(m \oplus G(r)))$. (If $x \notin D$ go to step 1.)
3. Compute encryption $c = f(x)$ – length of x and of c is n.

Decryption of a cryptotext c.

- Compute $f^{-1}(c) = a\|b$, $|a| = l$ and $|b| = k$.
- Set $r = h(a) \oplus b$ and get $m = a \oplus G(r)$.

<u>Comment</u> Operation "||" stands for a concatenation of strings.

Cases when RSA is easy to break

- If a user U wants to broadcast a value x to n other users, using for a communication with a user P_i a public key (e, N_i), where e is small, by sending $y_i = x^e \bmod N_i$

Low exponent attacks:

- If e = 3 and 2/3 of the bits of the plaintext are known, then one can decrypt efficiently
- If two plaintexts differ only in a (known) window of length 1/9 of the full length and e = 3, one can decrypt the two corresponding cryptotexts
- Wiener showed how to get secret key efficiently if $d < 1/3\, N^{1/4}$

RSA Standards

PKCS (public-key Cryptography Standards) is a set of algorithms published by the RSA Data Security company. One of them is PKCS#1v2.1 - a modification of randomized RSA.

Let modulus n have k bytes, algorithm will encrypt messages m of length at most k - 11 bytes.

Encryption:

- Generate a pseudorandom string PS such that m and PS have total length k - 3 bytes
- Create byte string 00||02||PS||00||m, where 0i is the byte representing i
- Use RSA to encrypt the integer version of the previous string and convert the result into a k byte string

Decryption:

- Convert the cryptotext into an integer and reject it if it is greater than modulus or k < 11
- Convert ciphertext to integer representation, perform RSA decryption and convert the result to byte string
- Check that string has form 00||02||PS||00||m for some PS that has no zero bytes
- The resulting m is plaintext

Side-channel attacks on cryptosystems

Another “cryptosystems attack philosophy” is to attack their physical implementations, i.e. the devices on which the cryptographic protocols are implemented.

Since crypto-protocols descriptions say a prior nothing about how protocols should be physically carried out over some physical devices, theoretical security proofs, even though they remain totally valid, do not provide any security guarantee against attacks made via physical side-channels, such as electromagnetic radiation, heat dissipation, noise, observation of computation time, power assumption, ...

There are two basic types of attacks:

- **Passive side-channel attacks**, also known as ”information leakage attacks”. Such attacks do not require to actively manipulate the computation, but only to monitor the side-channel leakage during the computation.
- **Active side-channel attacks**, in which we assume that the attacker actively manipulates the execution of cryptographic algorithm (trying for example to introduce faults in the computation).

Attacks on RSA implementations

In 1995, Paul Kocher, an undergraduate of Stanford, discovered that Eve could recover decryption exponent by counting time (energy consumption) needed for exponentiation during several decryptions. The point is that if $d = d_k d_{k-1} \ldots d_1$, then at the computation of c^d, in the i-th iteration, a multiplication is performed only if $d_i = 1$ (and that requires time and energy).

STREAM CRYPTOSYSTEMS

A *stream cryptosystem* encrypts a stream of plaintext on the fly.

Stream cryptosystems are of large practical importance.

Most of the stream cryptosystems use one-time pad for encryption and differ in the way (pseudo)-random key-stream is generated.

Two basic key-stream generation techniques are:

- using a pseudorandom-generator
- using a finite automaton

Encryption is done either bit-wise or byte-wise.

RC4 STREAM CRYPTOSYSTEM

RC4 was designed by R. Rivest in 1987 and kept as a commercial secret till 1994. Some internet browsers/servers use RC4.

RC4 works as a finite automaton with an internal state. Its initial state is derived from the secret key only. Its internal state and next byte of the plaintext determine its next internal state and a new byte of the cryptotext, by making XOR of last bytes of plaintext and key.

The internal state consists of a triple (i, j, S), where i and j are bytes and S is a permutation on the set

$$\{0, 1, ..., 255\}$$

of bytes and it is encoded as an array S[0], S[1], ..., S[255].

Key is represented as an array

$$K[0], K[1], ..., K[keylength - 1]$$

of bytes.

RC4 STREAM CRYPTOSYSTEM (cont.)

The initial state is designed as follows:

$j \leftarrow 0;$

for $i = 0$ **to** 255 **do** $S[i] \leftarrow i;$

for $i = 0$ **to** 225 **do**

$j \leftarrow j + S[i] + K[i \bmod keylengthl] \bmod 256;$

$swap(S[i], S[j]);$

Plaintexts are iteratively encrypted and the initial state for a new plaintext is equal to the final state of the previous plaintext.

$i \leftarrow 0; j \leftarrow 0;$

Key-stream generator:

i $\leftarrow$ (i + 1) mod 256; j $\leftarrow$ (j + S[i]) mod 256;

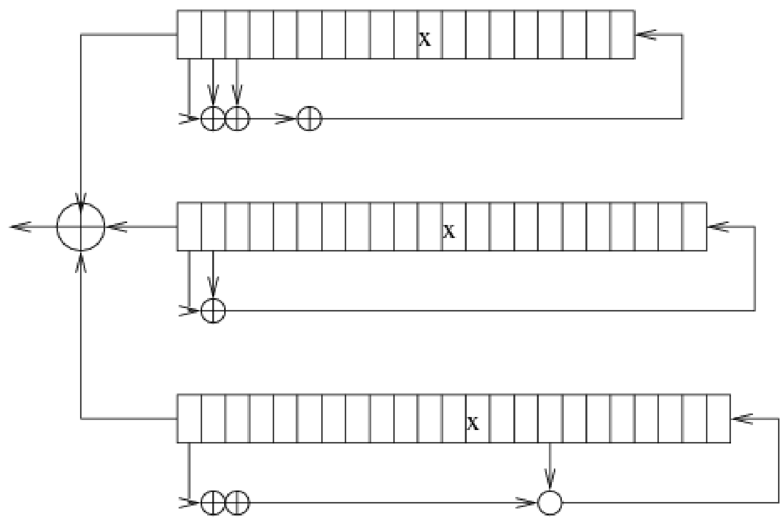
swap (S[i], S[j]);

output S[S[i] + S[j]] mod 256;

A5/1 – GSM encryption

A5/1 is used in the GSM mobile telephone networks. The description of A5/1 was secret, but it was reverse engineered and published on Internet.

A5/1 is based on a FA *A* that is based on the following three LFSRs (linear feedback shift registers) with a mutual shift control.



Three registers R_1, R_2 and R_3, contain 19 + 22 + 23 = 64 bits. Every time unit some of the registers is shifted - that is its content is shifted by one position and one new bit is pushed in. The new bit is the XOR of a few bits of the three LFSRs involved.

A5/1 – GSM encryption (cont.)

At each step those registers are shifted that have in a special cell, denoted by *x*, such a bit that is in the majority of bits of all three special cells.

Initiation phase (that uses a 64-bit secret key register K):

1: set all registers to zero;

2. **for** i = 0 **to** 63 **do**

   $R_1[0] \leftarrow R_1[0] \oplus key[i];$

   $R_2[0] \leftarrow R_2[0] \oplus key[i];$

   $R_3[0] \leftarrow R_3[0] \oplus key[i];$

   shift all registers;

3. **for** i = 0 **to** 21 **do**

   $R_1[0] \leftarrow R_1[0] \oplus count[i];$

   $R_2[0] \leftarrow R_2[0] \oplus count[i];$

   $R_3[0] \leftarrow R_3[0] \oplus count[i];$

   shift all registers;

4. **for** i = 0 **to** 99 **do** shift the automaton

where “count” is a 22-bit registers that counts “frames” of the plaintexts, where each frame has 114 bits.

All that corresponds to 4 hours of GSM communication.

SYMMETRIC CRYPTOSYSTEMS - BRUTE FORCE ATTACKS

We will discuss several types of brute force attacks that can be applied to any symmetric cryptosystem C_k considered as an oracle that for each given key as input replies whether it is a correct key.

Exhaustive search

This method consists of trying all possible keys exhaustively until the correct key is found. Exhaustive search can be made more efficient if a probability distribution on keys can be guessed or keys are known to satisfy some relations.

Dictionary attack

Creation of dictionary: For a fixed *x* and many *k* values $C_k(x)$ are computed and pairs $(C_k(x), k)$ are inserted into dictionary that is ordered according first item of each pair.

Search: If we obtain a $C_k(x)$ value (by chosen plaintext attack), dictionary gives us a list of potential keys.

A generalization of searching for several keys having several values $C_k(x)$ is easy.

Hellman's method

This method (suitable for the chosen plaintext attack) speeds up exhaustive search using large pre-computed tables and making time-memory tradeoff.

Method assumes that all encryptions of a given plaintext x have the same size, larger than the key length. The methods uses various (random) “reduction functions” R_l, that map cryptotext to strings of the key length, and functions

$$f_s(k) = R_s(C_k(x))$$

to compute, using iteration

$$k_{s,i,j} = f_s(k_{i,j-1})$$

(for a chosen l, m and s = 1, ..., l ; i = 1, ... , m; j = 1, …, t and randomly chosen $k_{s,i,0}$) values $k_{s,i,t}$ to get triplets (s, $k_{s,i,t}$, $k_{s,i,1}$).

Attack for an input y = $C_k(x)$:

for s = 1 **to** l **do**
  $j \leftarrow 1$; $k \leftarrow R_s(y)$;
  while there is no (s, k, .) entry and j <= t **do**
    $j \leftarrow j + 1$; $k \leftarrow f_s(k)$;
  if there is an (s, k, .) entry (s, k, k’) **then**
    while $C_{k'}(x) \neq y$ and i <= t **do**
      $j \leftarrow j + 1$; $k' \leftarrow f_s(k')$;
  if $C_{k'}(x) = y$ **then output**(k’);
otherwise the attack failed.

Secure communication in practice

Secure communication (session) between two parties usually proceeds by the following protocols:

- Protocol for parties (peer) identification.
- Exchange of the public-key material.
- Authenticated key generation protocol (and the resulting key is divided into several subkeys).
- Message security (integrity, authentication, confidentiality) is ensured by means of MAC and encryption protocols.

Some additional security requirements:

- To ensure proper sequentiality of messages (usually done by means of a synchronized message counter).
- Timeliness of message delivery (in time).
- Termination fairness: parties should be ensured to terminate the session in the same state.
- Anonymity (of parties should not leak out).
- Untraceability (of parties in later sessions).

SSH - Secure SHell protocol

SSH is to enable secure remote access to a computer - to implement secure (i.e. confidential and authenticated) communication channel in a client-server session.

When a client wishes to connect to a server, the server sends its public-key together with a certificate (if available).

Either client is able to authenticate the public key or the client has to trust that the public key is correct. The client then stores the public key in a file that has integrity protection.

If the above first connection is OK, then all future connections to the same server should be secure by comparing the received key with the stored key.

If keys do not match, the user gets a security warning (that can be ignored).

A commitment scheme based on discr. log.

Alice commits herself to an $m \in \{0,\ldots,q-1\}$.

Scheme setting:

Bob randomly chooses primes p and q such that

$$q \mid (p-1).$$

Bob chooses random generators $g \neq 1 \neq v$ of the subgroup G of order $q \in Z_p^*$.

Bob sends p, q, g and v to Alice.

Commitment phase:

To commit to an $m \in \{0,\ldots,q-1\}$, Alice chooses a random $r \in \{0,\ldots,q-1\}$, and sends $c = g^r v^m$ to Bob.

Opening phase:

Alice sends r and m to Bob who then verifies whether $c = g^r v^m$.

COMMITMENTS and ELECTRONIC VOTING

Let com$(r, m) = g^r v^m$ denote commitment to m in the commitment scheme based on discrete logarithm. If $r_1, r_2, m_1, m_2 \in \{0,\ldots,q-1\}$, then

$$\text{com}(r_1, m_1) \times \text{com}(r_2, m_2) = \text{com}(r_1 + r_2, m_1 + m_2).$$

Commitment schemes with such a property are called homomorphic commitment schemes.

Homomorphic schemes can be use to cast yes-no votes of n voters $V_1,\ldots, V_n$, by the trusted authority TA for whom e_T and d_T are ElGamal encryption and decryption algorithms.

Each voter V_i chooses his vote $m_i \in \{0,1\}$, a random $r_i \in \{0,\ldots, q-1\}$ and computes his voting commitment c_i = com(r_i, m_i). Then V_i makes c_i public and sends $e_T(g^{r_i})$ to TA and TA computes

$$d_T\left(\prod_{i=1}^{n} e_T\left(g^{r_i}\right)\right) = \prod_{i=1}^{n} g^{r_i} = g^r,$$

where $r = \sum_{i=1}^{n} r_i$ and makes public g^r.

Now, anybody can compute the result s of voting from publicly known c_i and g^r since

$$v^s = \frac{\prod_{i=1}^{n} c_i}{g^r},$$

with $s = \sum_{i=1}^{n} m_i$.

s can now be derived from v^s by computing $v^1, v^2, v^3,\ldots$ and comparing with v^s if the number of voters is not too large.

Voting Protocols – Advanced Settings

- In voting protocols we have a set $V = \{v_1, . . . , v_n\}$ of voters and a set $A = \{a_1, . . . , a_m\}$ of election authorities
- Communication is through a communication channel with memory called *bulletin board*. Each subject can write to his part of the bulletin board any message and that can be read by anyone.
- Electronic voting schemes are clearly ways to go. However, it is not easy to make them to be sufficiently reliable.
- A voting protocol specifies to voters and authorities how they should behave:
  a) before voting (initialization phase)
  b) during voting
  c) after voting (counting of votes phase)

Basic Requirements on Voting Protocols

- Only legitimate voters can vote and each only once.
- There is a security parameter t, such that no group of t voters not containing a voter v_i and at most $t - 1$ voting authorities, can determine the vote of v_i.
- Each voter can verify whether his vote was counted
- Anyone can verify the final result of elections .
- There is a t_0 such that the system can manage incorrect behavior of any group of voters and at most $t_0 - 1$ voting authorities.
- No voters is able to prove how (s)he voted .

SECURE ELECTIONS

Another set of properties of voting protocol:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can change anyone else vote without being discovered.
5. All voters can make sure that their votes were counted.

Additional requirement: Everyone knows who voted and who didn't.

Very simple voting protocol I.

- All voters encrypt their vote with the public key of a Central Election Board (CEB).
- All voters send their votes to the CEB.
- CEB decrypts votes, tabulates them and makes the result public.

The protocol has problem with some of the required properties.

Simple voting protocol II.

- Each voter V_i signs his/her vote v_i with his/her private key – $d_{V_i}(v_i)$.
- Each voter encrypts his/her signed vote with the CEB's public key – $e_{CEB}(d_{V_i}(v_i))$.
- All voters send their votes to CEB.
- CEB decrypts the votes, verifies signatures, tabulates votes and makes the result public.

Voting protocol (Nurmi, Salomaa, Santean)

- CEB publishes a list of all legitimate voters.
- Within a given deadline, everybody intended to vote reports his/her intention to CEB.
- CEB publishes a list of voters participating in elections.
- Each voter V receives an identification number, i, using a special protocol that very likely assigns different numbers to different users.
- Each voter V creates a public encryption function e_V and secret decryption function d_V.
- *If v is a vote of the voter V, then V generates the following message and sends it to CEB:

$$(i, e_V(i, v))$$

- The CEB acknowledges the receipt of the vote by publishing $e_V(i, v)$.
- Each voter V sends to CEB the pair (i_V, d_V).
- The CEB uses d_V to decrypt the vote $(i, e_V(i, v))$.
- At the end of the elections CEB publishes the results of the election and, for each different vote, the list of all $e_V(i, v)$ - values that contained that vote.
- It is possible that two voters get the same identification number. In such a case, the
- CEB generates a new identification number, i_1, chooses one of two votes, and publishes: $(i_1, e_V(i, v))$. The owner of that vote recognizes that and sends in a second vote, repeating step (*) with the new identification number i_1.

Anonymous money order

Digital cash idea has one big problem: how to hide to whom you gave the money.

Protocol 1

(1) Alice prepares 100 anonymous money order for 1000$.

(2) Alice puts one money order, and a piece of carbon paper, into each of 100 different envelopes and gives them to the bank.

(3) The bank opens 99 envelopes and confirms that each is a money order for 1000$.

(4) The bank signs the remaining unopened envelope. The signature goes through the carbon paper to the money order. The bank hands the unopened envelope back to Alice and deletes 1000$ from her account.

(5) Alice opens the envelope and spends the money order with a merchant.

(6) The merchant checks for the bank's signature to make sure the money order is legitimate.

(7) The merchant takes the money order to the bank.

(8) The bank verifies its signature and credits $1000 to the merchnt's account.

(Alice has a 1% chance of cheating - the bank can make penalty for cheating so large that this does not pay of.)

ANONYMITY problems

Very often it is of importance for a party involved in an information transmittion process that its identity remains hidden.

There is a variety of problems that require that a communicating party remains hidden or anonymous.

For example, anonymous broadcast is a process P that has one anonymous sender and all other parties in communication receive the message m that has been sent by A.

Another example of anonymity in communication is so-called anonymous many-to-one communication at which all parties send their messages and there is only on receiver

Anonymous transfer protocols

- The term anonymous transfer includes a variety of different tasks.
- Anonymity of an object is the state of being not identifiable with any particular element of a set of subjects known as an anonymity set.
- An anonymity set consists of a set P of participants able to perform a particular action we are interested in. (For example, that a real sender (receiver) is not identifiable within a set of potential senders (receivers)).

* Cheating is usually modeled by an adversary A not in P, who has a full control of some subset M of P of (malicious) participants. (A is assumed to have access to memories, inputs and outputs of all participants from M – this way one can model the case malicious participants cooperate.)

Chaum's anonymous broadcast

Let a communicating scheme be modeled by an unoriented graph G= (V,E),
With V={1,2,…,n}, representing nodes (parties) and E edges (communication links).

PROTOCOL: Each party P_i performs (all in parallel) the following actions:

- For each $j \in \{1,2,\ldots,n\}$ it sets $k_{ij} = 0$;
- If $(i, j) \in E$, $i < j$, randomly chooses a key k_{ij} and sends it securely to P_j;
- If $(i, j) \in E$, $j < i$, after receiving k_{ij} it sets $k_{ij} = - k_{ij} \bmod n$;
- P_i broadcasts $O_i=m_i+\Sigma_j k_{ij} \bmod n$, where $m_i \in \{0,1,\ldots,n-1\}$ is the message being sent by P_i;
- P_i computes the global sum $S = \Sigma_j O_j \bmod n$.

- Clearly, $S=\Sigma_j m_j \bmod n$, and therefore if only one $m_j \neq 0$, all participants get that message.

- One can show that to preserve anonymity of a correctly behaving sender P_i,
  It is sufficient that one another participant P_j such that $(i,j) \in E$ behaves correctly.

PRIVACY PRESERVATION

PROBLEM: An important problem is whether and how we can build a statistical database D of important information about a population P so that privacy of individuals of P is preserved.

Can we define perfect privacy in the following way that would be analogical to the perfect semantical security of encryptions: Nothing about an individual of P should be learnable from the database that could not be learned without the access to the database.

ANSWER: NO

SOLUTION: Differential privacy: The risk to one's privacy, or in general, any type of the risk, should not substantially increase as the result of participation in the statistical database.

EXAMPLE

The reason why the ideal privacy, namely that the access to a statistical database should not enable one to learn anything about an individual that could not be learned without access,

is not achievable,

is due to the fact that an auxiliary information can be available from the database to the adversary.

For example, let us assume that we have a statistical database of heights of women of different nationalities in Asia and the auxiliary information that Madonna is 3 cm higher than an average women in Pakistan.

That would provide a potentially sensitive information about Madonna, in spite of the fact that she did not participate at the creation of the above mentioned database.

DINING CRYPTOGRAPHERS

- Three cryptographers have dinner at a round table of a 5-star restaurant.
- Their waiter tells them that an arrangement has been made that their bill for dinner will be paid anonymously – either by one of them, or by NSA.
- Cryptographers respect each other's right to make anonymous payment, but they would like to know whether payment was done by NSA.
- Is there a way for them to learn whether one of them paid the bill without knowing which one (for other two)?

PROTOCOL for CRYPTOGRAPHERS

PROTOCOL:

- Each cryptographer flips a perfect coin between him and the cryptographer on his right, so that only two of them can see the outcome.
- Each cryptographer who did not pay the bill states aloud whether the two coins he see – the one he flipped and the one his right-hand neighbor flipped – fell on the same side or on different sides.
- The cryptographer who paid the bill states aloud the opposite he sees.

CORRECTNESS:

- An odd number of differences claimed by cryptographers implies that a cryptographer paid the bill.
- An even number implies that NSA paid the bill.
- In case a cryptographer paid the bill the other two will have no idea he did.

Secure contract signing protocol

Alice and Bob want to sign a contract *C*. They will use a SKC *S* and an 1-2 OT as follows.

- Alice and Bob, independently and randomly, select each a set of n keys for *S*

$$\{(l_j^A, r_j^A)\}^n_{j=1} \quad \{(l_j^B, r_j^B)\}^n_{j=1}$$

- Alice and Bob, independently, generate *n* signatures of *C*

$$\{S_j^A=(L_j^A, R_j^A)\}^n_{j=1} \quad \{S_j^B= (L_j^B, R_j^B)\}^n_{j=1}$$

where L_j^X and R_j^X, for $X \in \{A,B\}$ are let and right halves of their respective signatures. Each S_j^X is assumed to be accompanied by a time stamp. (The contract will be considered to be signed if all L_j^X and R_j^X can be produced by each of the parties.)

Secure contract signing protocol II

- Alice and Bob, independently, encrypt each signature as follows

$$\{(l_j^A(L_j^A),\ r_j^A(R_j^A))\}^n_{j=1} \qquad \{(l_j^B(L_j^B),\ r_j^B(R_j^B))\}^n_{j=1}$$

  and they send, to each other, their respective pairs of the encrypted signatures.
- Using 1-2 OT, Alice and Bob send to each other exactly one of their keys (l_i^X , r_i^X) for all i, so neither of them knows which half they got.
- Alice and Bob, independently, decrypt which messages they can, ensuring they do indeed have a legitimate message in each case.
- Alice and Bob alternate in sending bits of their $2n$ keys, until all verifying bits have been received by both of them. Once this is done each of them can decrypt second half of the corresponding message and contract is signed.

Key agreement and authentication

- A variety of protocols have been developed to connect hosts on Internet. (Host are here those computers that provide services to other computers and users of Internet.)
- TCP/IP (Transmission Control Protocol/Internet protocol) is a set of communication protocols used to connect hosts on Internet.
- Important protocols are EKE (Encrypted Key Exchange patented in 1993) and SPEKE (Simple Password Exponential Key Exchange) and their various modifications.
- Of large importance is Secure Remote Protocol (SRP-6). In this protocol Alice interacts with Bob to establish a password k, and upon mutual authentication, a session key S is derived that is then used to establish a ”permanent” key, to be used to encrypt all future traffic.

SRP-6

Public values: A large prime p is chosen, such that $(p - 1)/2$ is also prime, a primitive root α modulo p and a hash function h. Protocol:

1. To establish a password k with Bob, Alice picks a salt s and computes $d = h(s, k)$, $v = \alpha^d \pmod p$. Bob stores v and s as Alice's password and salt.
2. Alice sends to Bob her identification I_a and $A = \alpha^a$, where a is a nonce.
3. Bob looks up Alice's password entry, retrieves v and s from her database and sends both s and $B = 3v + \alpha^b$, where b is another nonce, to Alice.
4. Alice and Bob compute, independently, $u = h(A,B)$.
5. Alice computes $S = (B - 3\alpha^d)^{(a+ud)}$. Bob independently computes $S = (Av^u)^b$.
6. Both, Alice and Bob compute $K = h(S)$.
7. To verify that she has the correct key, Alice sends to Bob
$$h_1 = h(h(p \oplus h(\alpha)), h(I_a), s, A, B, K).$$
8. Bob computes h_1, compares with value received from Alice and if they agree, he sends to Alice
$$h_2 = h(A, h_1, K).$$
9. Upon receiving h_2 Alice verifies that K is a correct key.

Digital cash transactions

Basic players and procedures:

Bank uses RSA with encryption (decryption) exponent e (d) and modulus n.

Digital money (m, m^d), where m is unique identification number of a coin, m^d is its bank signature. Bank records all coin identification numbers in a database of used coins together with an identification of the money owner.

Blind signatures - blinding To sign a coin m by a bank, customer (Bob) chooses a random r, sends t = $r^e m \pmod{n}$ to bank. the bank signs it and sends $u = t^d$ to Bob. By computing ur^{-1} Bob gets m^d.

Secret splitting (sharing) To split a binary-string secret s, a random r is chosen and s is split to r and $s \oplus r$.

E-cash withdraw

- Bob generates 100 sets of 100 unique strings $S_j = \{I_{j_k}\}_{k=1}^{100}$, $1 \leq j \leq 100$, such that each I_{j_k} uniquely identifies Bob.
- Bob splits each I_{j_k} into two pieces $I_{j_k} = (L_{j_k}, R_{j_k})$.
- Bob sends to bank 100 blinded money orders

$$M_j = (100\$, m_j, r_j^e m_j, \{L_{j_k}, R_{j_k}\}_{k=1}^{100}),$$

where all m_j and r_j are randomly chosen.
- Bank chooses randomly one of 100 money orders, say M_{100}, checks that all remaining ones are for the same amounts, have different m_j and that each $L_{j_k} \oplus R_{j_k}$ identifies Bob. If all is O.K. Bank signs M_j.
- Bob unblinds signature to get ECash coin (m_{100}, m_{100}^d).

E-cash spending

1. Shop verifies bank's signature by computing $(m_{100}{}^{d})^{e} = m_{100}$.

2. Shop sends Bob a random binary string $b_1 b_2 \ldots b_{100}$ and asks Bob to reveal L_{100_k} if $b_k = 1$ and R_{100_k} if $b_k = 0$ what Bob does, for all k.
   Afterwards, shop sends the money order to bank together with the chosen binary string b and Bob's responses.

3. Bank checks its used coins database. If m_{100} is not there, bank deposits 100\$ into shop's account and m_{100} into its used coins database, together with Bob's identification, and let shop to know that the money order is O.K. Shop then sends goods to Bob.

E-cash spending II

4. If m_{100} is in the database of used coins, the money order is rejected. Bank then compares the identity string on false money order with the stored identity string attached to m_{100}. If they are the same, bank knows that shop duplicated the money order. If they differ, then bank knows that the entity who gave it to the shop must have copied it.

   In case the coin (m_{100}, m_{100}^d). was spent with another shop, then that shop gave Bob another binary string (in step 2). Bank compares corresponding binary strings to find an i, where i-th bits differ. This means that one shop asked Bob to reveal R_i and second L_i. By computing $L_i \oplus R_i$ bank reveals Bob's identity, which can be reported to authorities.