LECTURE NOTES ON PART II OF INTRODUCTION TO CRYPTOGRAPHY

COURSE: COT6421, SPRING 1998
INSTRUCTOR: DR. RAKESH SINHA
PRESENTER: KONSTANTIN BEZNOSOV

1. Probabilistic Encryption[1]

The notion of probabilistic encryption was invented by Shafi Goldwasser and Silvio Micali [2] in 1982.

1.1. **Why do we need probabilistic encryption?**

The point of probabilistic encryption is to eliminate any information leaked with public-key cryptography. Assuming a cryptanalyst has a ciphertext $C = E_k(M)$ of message M encrypted with public key E_k and he wants to recover message M. If he guesses correctly a random message M' such that $C' = E_k(M')$ and C'=C, then he knows that M'=M. If it's wrong, he just guesses again. so, there is potential problem to allow a cryptanalyst to encrypt random messages with your public key. Probabilistic encryption tries to eliminate that leakage.

1.2. **The idea behind probabilistic encryption.**

The particular ciphertext used in any given encryption is randomly chosen.

$$C_1 = E_k(M),\, C_2 = E_k(M),\, C_3 = E_k(M), ...,\, C_i = E_k(M)$$

A large number of ciphertexts will decrypt to a given plaintext.

$$M = D_k(C_1) = D_k(C_2) = D_k(C_3) = \ldots D_k(C_i)$$

We will show shortly how a ciphertext can be decrypted with the private key. Assume the cryptanalyst has ciphertext $C_i = E_k(M)$. Even if he guesses M correctly, when he encrypts $E_k(M)$, the result C_j will be most probably a completely different from C_i. He cannot compare C_i and C_j, and so cannot know that he has guessed the message correctly. Even if the cryptanalyst has the public key, the plain text, and the ciphertext, he cannot prove that the ciphertext is the encryption of the plaintext without the private decryption key.

1.3. **Simplified description of the optimized algorithm.**

Two years later after the original work, Blum and Goldwasser had an efficient implementation of probabilistic encryption using the Blum Blum Shub (BBS) random-bit generator[2]. The BBS generator is based on the theory of quadratic residues. In English, there are two primes, p and q, such that they are congruent[3] to 3 modulo 4. numbers p and

[1]This section material is adopted from [1, pages 552-554].

[2]The most important information to know about BBS generator is that:

1. You do not have to iterate through all i-1 bits to get the ith bit.
2. For a given sequence generated by the generator, a cryptoanalyst cannot predict the next bit in the sequence nor the previous bit in the sequence.

[3]For those who cannot remember (like me): Number a is called CONGRUENT to b, modulo n, if $a \equiv b + kn$ for some integer k. Short form is $a \equiv b (mod\, n)$.

q are the private key. Their product, $pq = n$, is the public key. As we know, the difficulty to find the private key from the public key is due to the difficulty of factoring n.

To encrypt a message M:

1. Choose some random x, relatively prime to n.
2. Compute $x_0 = x^2 \, mod \, n$
3. Run BBS generator with x_0 as the seed. The generator spits out bits b_i, where each b_i is the least significant bit of $x_i \equiv x_{i-1}^2 mod \, n$
4. Use the output of the generator as a stream cipher.
5. Compute XOR M, one bit at a time, with the output of the generator.
   M=$m_1, m_2, m_3, ..., m_t$
   C=$m_1 \oplus b_1, m_2 \oplus b_2, m_3 \oplus b_3, ..., m_t \oplus b_t$
6. Append the last computed value, x_t, to the end of the message C. Lastly generated at step 3 number x_t is needed for recovering x_0 by using the private key.

In order to decrypt the ciphertext with the private key, one needs to recover x_0. Values of p,q,n, t and x_t are used for it. Once we have x_0, we decrypt the ciphertext C by setting up the BBS generator and XORing the output with the ciphertext. It can be proved that the difficulty of breaking the this scheme is the same as difficulty of factoring n.

1.4. **Drawbacks of probabilistic encryption.** Due to the fact that many ciphertexts decrypt to the same plaintexts, the ciphertext will always be larger than the plain text. The implementation by the original authors [2] of probabilistic encryption would produce ciphertext so much larger than the plain text that it was unusable.

Also, this scheme is totally insecure against a chosen-ciphertext attack[4]. From the least significant bits of the right quadratic residues (see the simplified probabilistic encryption algorithm above), it is possible to calculate the square root of any quadratic residue. If we can do this, then we can factor n.

1.5. **Relation Between Average Case Computational Difficulty and the Worst Case Difficulty.** Ajtai and Dwork [3] presents a public key cryptosystem generator with the property that if a random instance of the cryptosystem can be broken than the worst-case unique shortest path vector (SPV) problem has probabilistic polynomial time solution. They claims this is the only public key cryptosystem with the property that to break a random instance is as hard as to solve the worst-case instance of the problem on which the system is based.

1.5.1. *Ideas Behind it.* An instance of the cryptosystem is a collection of *hidden hyperplanes*[5], which form the PRIVATE KEY, together with a method of generating a point guaranteed to be near one of the hyperplanes in the collection. The PUBLIC KEY is a set of random points near the hyperplanes induced by a random vector z in the n-dimensional unit ball. The public key is chosen so as not to reveal the collection of hyperplanes. the sum of a random subset of these points is itself close to a hyperplane. A large n-dimensional cube $\vartheta \in R^n$is selected and fixed.

[4]When a cryptoanalyst not only has access to the ciphertext and associated plaintext for several messages, but also chooses the plaintext that gets encrypted.

[5]A hyperplane is a part of a vector space. It is "flat" (the official name is "affine") in the sense that if you pick any two different points in it, the straight line passing through them (extended beyond them, too) will all lie in it. Among other affine sets it is distinguished by having dimension one less than the dimension of the whole space (finite dimension of the surrounding space is assumed). In a 3-dimensional space, hyperplanes are 2-dimensional, and are better known as "planes". In a 2-dimensional space, hyperplanes are 1-dimensional, known as "straight lines".

Encryption is bit-by-bit:

- zero is encrypted by using public key to find a random vector $v \in \vartheta$ near one of the hyperplanes – the ciphertext is v;
- one is encrypted by choosing a random vector u uniformly from ϑ – the ciphertext is simply u.

Decryption of a ciphertext x is performed using the private key to determine the distance of x to the nearest hidden hyperplane. If this distance is sufficiently small, then x is decrypted as zero; otherwise x is decrypted as one. There is a small (but polynomial) probability of an error in decryption: an encryption of one may be decrypted as zero.

The ability to distinguish encryptions of zero from encryptions of one yields the ability to solve the hidden hyperplane problem. This implies that the only way to break the cryptosystem is to find the private key.

1.5.2. *The Main Theorem: Worst-Case/Average-Case Equivalence.*

Theorem. *For all $c_1, c_2, c_3, c_4 > 0$ there exists a c_5 and a probabilistic algorithm B (using an oracle) so that for all sufficiently large n, condition (1) implies condition (2), where*

1. *A is a probabilistic circuit size n^{c_1} so that if $u, v_1, ..., v_m$ are picked at random as described in the protocol generating the public and private keys, then with probability of at least n^{-c_2}, A distinguishes the random variables $S_{v_1,...,v_m}$ and $E_{v_1,...,v_m}$, given $v_1, ..., v_m$ with probability at least $\frac{1}{2} + n^{-c_3}$.*
2. *B, using A as an oracle, can solve any instance of size at most n^{c_4} of the n^{D_2}-unique shortest vector problem in time n^{c_5} and with a probability at least $1 - 2^{-n}$.*

The theorem proof is beyond the scope of this presentation.

2. Identity-Based Public-Key Cryptography[6]

Consider the following scenario: Alice wants to send a secure message to Bob. She does not want to get public key from a key server; she doesn't want to verify some trusted third party's signature on his public-key certificate; and she doesn't even want to store Bob's public key on her own computer. She just wants to send him a message.

Identity-based public-key cryptosystems, sometimes called Non-Interactive Key Sharing systems, solve this problem. With identity-based cryptography, Bob's public key is his identity. It is a ideal system as it could be about sending secure e-mail messages. It makes the cryptography about as transparent as possible.

The system is based on Trent issuing private keys to users based on their identity. If Alice's private key is compromised she has to change some aspect of her identity to get another one. A serious problem is designing a system in such a way that a collusion of dishonest users cannot forge a key.

Unfortunately, most solutions turned out to be insecure or are very complicated to be useful in real life. Many of the proposed solutions involve Trent choosing a random number for each user – this defeats the real point of the system. Nothing proposed so far is both practical and secure.

[6]This section material is adopted from [1, page 115].

3. Fair Coin Flipping Using Public-Key Cryptography[7]

The protocol works using either public-key cryptography or symmetric cryptography. The only requirement is that the algorithm commute, i.e.

$$D_{K_1}(E_{K_2}(E_{K_1}(M))) = E_{K_2}(M)$$

In general, this property is not true for symmetric algorithms, but it is true for some public-key algorithms. For example, RSA with identical moduli.

Here is the protocol:

1. Alice and Bob each generate a public-key/private-key key pair.
2. Alice generates 2 messages M_T – indicating heads – and M_H – indicating tails. Each message contains some unique random string that will allow Alice to distinguish M_T from M_H later. Alice encrypts both messages with her public key and sends them – $E_A(M_T), E_A(M_H)$ – in random order.
3. Bob chooses arbitrary one of those messages when he receives them. He encrypts the message with his public key and sends it back to Alice.
$$E_B(E_A(M))$$
where M is either M_T or M_H.
4. Alice, who cannot read the message sent back to her, decrypts it with her private key and then sends it back to Bob.
$$D_A(E_B(E_A(M))) = E_B(M)$$
5. Bob decrypts the message with his private key to reveal the result of the coin flip. He sends the decrypted message to Alice.
$$D_B(E_B(M)) = M$$
where M is either M_T or M_H.
6. Alice reads the result of the coin flip and verifies that the arbitrary string (either for M_T or M_H) is correct.
7. Both Alice and Bob reveal their key pairs so that both can verify that the other did not cheat.

A real application for this protocol is session-key generation. Coin-flipping protocols allow Alice and Bob to generate a random session key such that neither can influence what the session key will be.

4. Fair Cryptosystems (Key Escrow)[8]

Simplistic scenario:

1. Alice creates her private-key/public-key key pair. She splits the private key into several public pieces and private pieces.
2. Alice sends a public piece and corresponding private piece to each of the trustees. These messages must be encrypted. She also sends the public key to the key distribution center (KDC).

[7]This section material is adopted from [1, pages 89-92].
[8]This section material is adopted from [1, pages 97-100, 546-548].

3. Each trustee, independently, performs a calculation on its public piece and its private piece to confirm that they are correct. Each trustee stores the private piece somewhere secure and sends the public piece to the KDC.
4. The KDC performs another calculation on the public pieces and the public key. Assuming that everything is correct, it signs the public key.

If the courts order a wiretap, then each of the trustees surrends their piece to the KDC, and the KDC can reconstruct the private key.

Fair Diffie-Hellman Algorithm. In the basic Diffie-Hellman algorithm, a group of users share a prime p, and a generator, g. Alice's private key is s, and her public key is $t = g^s \, mod p$.

More detailed example:

1. Alice chooses five integers, $s_1, s_2, s_3, s_4 \, and \, s_5$, each less than p-1. Alice's private key is
$$s = \{s_1 + s_2 + s_3 + s_4 + s_5\} \, mod p - 1$$
and her public key is
$$t = g^s \, mod p$$
Alice also computes $t_i = g^{s_i} \, mod p, \, for \, i = 1 \, to \, 5$ Alice's public shares are t_i, and her private shares are s_i.
2. Alice sends a private piece and corresponding public piece to each trustee. She sends t to KDC.
3. Each trustee verifies that
$$t_i = g^{s_1} \, mod p$$
If it does, the trustee signs t_i and sends it to the KDC. The trustee stores s_i in a secure place.
4. After receiving all five public pieces, the KDC verifies that
$$t = \{t_1 * t_2 * t_3 * t_4 * t_5\} mod p$$
If it does, the KDC approves the public key.

At this point, the KDC knows that the trustees each have a valid piece and that they can reconstruct the private key if required. Neither KDC nor any of four of the trustees working together can reconstruct Alice's private key.

Research in this area found how to make RCA fair and how to combine a threshold scheme with the fair cryptosystem, so that m out of n trustees can reconstruct the private key.

5. Zero Knowledge Interactive Proof Systems("ZKIPS")[9]

The general question is whether it is possible to prove a statement without yielding anything beyond its validity. Such proofs, whenever they exist, are called ZERO-KNOWLEDGE. Loosely speaking, zero-knowledge proofs are proofs that yield nothing (i.e., "no knowledge") beyond the validity of the assertion.

In "canonical" mathematics proofs have a static nature (e.g., they are "written"), whereas in real-life situations proofs have a dynamic nature (i .e., they are established via an interaction). The dynamic interpretation of the notion of a proof is more adequate to our setting in which proofs are used as tools (i.e., subprotocols) inside "cryptographic" protocols. Furthermore, the dynamic interpretation (at least in a weak sense) is essential to the non-triviality of the notion of a zero-knowledge proof.

[9]This section material is adopted from[4] and from [5].

To give a better flavor of the definition, we now consider a conversation between Alice and Bob in which Bob asks Alice questions about a large graph (that is known to both of them). Consider first the case in which Bob asks Alice whether the graph is Eulerian or not. Clearly, we say that Bob gains no knowledge from Alice's answer, since he could have determined the answer easily by himself (e.g., by using Euler's Theorem which asserts that a graph is Eulerian if and only if all its vertices have even degree). On the other hand, if Bob asks Alice whether the graph is Hamiltonian or not, and Alice (somehow) answers this question then we cannot say that Bob gained no knowledge (since we do not know of an efficient procedure by which Bob can determine the answer by himself, and assuming P$\neq$ NP no such efficient procedure exists). Hence, we say that Bob gained knowledge from the interaction if his computational ability, concerning the publicly known graph, has increased (i.e., if after the interaction he can easily compute something that he could not have efficiently computed before the interaction). On the other hand, if whatever Bob can efficiently compute about the graph after interacting with Alice, he can also efficiently compute by himself (from the graph) then we say that Bob gained no knowledge from the interaction. Hence, Bob gains knowledge only if he receives the result of a computation which is infeasible for Bob. The question of how could Alice conduct this infeasible computation (e.g. , answer Bob's question of whether the graph is Hamiltonian) has been ignored so far. Jumping ahead, we remark that Alice may be a mere abstraction or may be in possession of additional hints, that enables to efficiently conduct computations that are otherwise infeasible (and in particular are infeasible for Bob who does not have these hints). (Yet, these hints are not necessarily "information" in the information theoretic sense as they may be determined by the common input, but not efficiently computed from it.) We wish to stress that knowledge (as discussed above) is very different from information (in the sense of information theory). Knowledge is related to computational difficulty, whereas information is not. Consider the case in which Alice answers each question by flipping an unbiased coin and telling Bob the outcome. From an information theoretic point of view, Bob gets from Alice information concerning an event. However, we say that Bob gains no knowledge from Alice, since he can toss coins by himself.

5.1. **A Concrete Example.** Concider a concrete example of zero-knowledge interactive proof.

Alice wishes to prove to Bob that she knows some item of knowledge without actually giving Bob any of that knowledge. Let us first imagine that Alice claims she knows a "Hamiltonian cycle"[10] on a particular graph. The particular graph may be "registered" somewhere with Alice's claim that she – and only she, for reasons discuss later at the end – knows a Hamiltonian cycle for the graph. In a sense, this is her "proof of identity."

To make this example concrete, Alice is using this piece of knowledge as her PASSWORD to get into some system. She presents a map of 50 cities and some set of highways interconnecting them and says "I am who I say I am if and only if I know a Hamiltonian cycle for this graph."

The conventional (non zero knowledge) way to convey this knowledge is for Alice to simply SHOW the Hamiltonian cycle to Bob. This is how passwords are currently handled. Bob, and anybody else who is spying on the exchange, then knows the "secret," which isn't a secret anymore.[11]

5.2. **Interactions During the Proof.** Alice, instead of showing Bob the Hamiltonian cycle, takes the cities and covers them with something, say, coins. (On a computer, this is all done in software, using the cryptographic protocol called "bit commitment.")

[10]For a given graph, a Hamiltonian cycle is one which passes through each node once and only once.
[11]Anybody who saw the exchange, including Sysadmin Bob, could then impersonate her.

Alice scrambles the position of the cities (covered by coins) so as not to allow positional cues.[12] Needless to say, she scrambles the cities out of sight of Bob, so he can't figure out which cities are which. However, once she's done with the scrambling, she displays the cities in such a way that she can't LATER CHANGE, .i.e., she "commits" to the values, using well-known cryptographic methods for this.

Bob sees 50 cities with links to other cities, but he doesn't have any way of knowing which of the covered cities are which. Nor, I should add, are the links labelled in any way–it wouldn't do to have some links permanently labelled "Route 66" or "Highway 101"!

She says to Bob: "Pick one choice. Either you can see a Hamiltonian cycle for this set of covered cities and links, or you can see the cities uncovered." In other words, "Alice cuts, Bob chooses."

Bob tosses a coin or chooses randomly somehow and says: "Show me the cities."

Alice uncovers all the cities and Bob examines the graph. He sees that Akron is indeed connected to Boise, to Chicago, to Denver, not to Erie, and so on. In short, he confirms that Alice has shown him the original graph. No substitution of another graph was made.

Bob, who is suspicious that this person is really who she claims to be, says to Alice: "Ok, big deal! So you anticipated I was going to ask you to show me the cities. Anybody could have gotten Alice's publicly registered graph and just shown it to me. You had a 50-50 chance of guessing which choice I'd make."

Alice smugly says to him: "Fine, let's do it again." She scrambles the cities (which are covered) and displays the graph to Bob – 50 covered cities and various links between them. She tells Bob to choose again.

This time Bob says: "Show me the Hamiltonian cycle."

Without uncovering the cities (which would give the secret away, of course), Alice connects the cities together in a legal Hamiltonian cycle.

Bob says, "OK, so this time you figured I was going to ask you the opposite of what I did last time and you just substituted some other graph that you happened to know the Hamiltonian cycle of. I have no guarantee the graphs are really the same."

Alice, who knows this is just the beginning, says: "Let's do the next round."

...and so it goes....

After 30 rounds, Alice has either produced a legal Hamiltonian cycle or a graph that is the same as (isomorphic to...same cities linked to same other cities) the registered graph in each and every one of the rounds.

There are two possibilities:

1. She's an imposter and has guessed correctly EACH TIME which choice Bob will make, thus allowing her to substitute either another graph altogether (for when Bob wants to see the Hamiltonian cycle) or just the original graph (for when Bob asks to see the cities uncovered to confirm it's the real graph). Remember, if Alice guesses wrong even once, she's caught red-handed.
2. She really is who she claims to be and she really does know a Hamiltonian cycle of the specified graph.

The odds of #1 being true drop rapidly as the number of rounds are increased, and after 30 rounds, are only 1 in 2^{30}, or 1 in a billion. Bob chooses to believe that Alice knows the solution.

[12]Most of the 50 cities should have about the same number, ideally exactly the same number, of links to other cities, to ensure that some cities are not "marked" by having some unique number of links. A detail.

Alice has conveyed to Bob proof that she is in possession of some knowledge without actually revealing any knowledge at all! The proof is "probabilistic."

This is the essence of a zero knowledge proof. There's more to it than just this example, of course, but this is the basic idea.

5.3. **Some Questions.**

1. Could someone else discover the Hamiltonian cycle of Alice's graph?

   Exhaustive search is the only way to guarantee a solution will be found–the Hamiltonian cycle problem is a famous "NP-complete" combinatorial problem. This is intractable for reasonable numbers of nodes. 50 nodes is intractable.
2. If finding a Hamiltonian cycle is intractable, how did Alice ever find one?

   She DIDN'T HAVE to find one! She started with 50 cities, quickly connected them so that the path went through each city only once and then wrote this path down as her "secret" solution. Then she went back and added the other randomly chosen interconnects to make the complete graph. For this graph, she obviously knows a Hamiltonian cycle, BY CONSTRUCTION.
3. Can Bob reconstruct what the Hamiltonian cycle must be by asking for enough rounds to be done?

   Not generally. Read the papers for details on this, which gets deeply into under what circumstance partial knowledge of the solution gives away the complete solution.

References

[1] Bruce Schneier. *Applied Cryptography, Second Edition: protocols, algorithms, and source code in C.* John Wiley & Sons, 2nd edition, 1996.

[2] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 365–377. ACM, May 1982.

[3] Miklos Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of Symposium on Theory of Computing*, pages 284–293. IBM, May 1997.

[4] Oded Goldreich. Foundations of cryptography (fragments of a book). http://www.wisdom.weizmann.ac.il/~oded/frag.html, 1995.

[5] Timothy C. May. Math: Zero knowledge proofs. Cypherpunks e-mail list, April 1993. http://www.best.com/~szabo/zkip.html.