0368.4162 Foundations of Cryptography December 12, 2017

Lecture 7: Public-Key Encryption

Lecturer: Nir Bitansky

1 Digital Signatures: Recap

Last time we saw how to construct one-time signature schemes for messages of arbitrary length, based on collision-resistant hash functions. In such schemes, given a signature on a single arbitrary message, it is hard to forge a signature on a new message. We then saw how this can be used to construct stateful signature schemes, where the size of signatures grows over time. Today, we'll make the last step and construct a stateless signature scheme, with signatures of fixed size.

The idea is to look at a full binary tree with nodes $X = \{x_1 \dots x_i \in \{0,1\}^i, 0 \leq i \in \mathbb{N}\}$ and associate for each node $x \in X$ a one-time pair (sk_x, vk_x). Then the signature on message $m \in \{0,1\}^*$ will include the chain of signatures from the root to the node $m \in X$. Of course that we cannot generate all pairs ahead of time — we will generate them using a PRF (to deal with messages of arbitrary length, we'll use a PRF for inputs of arbitrary length).

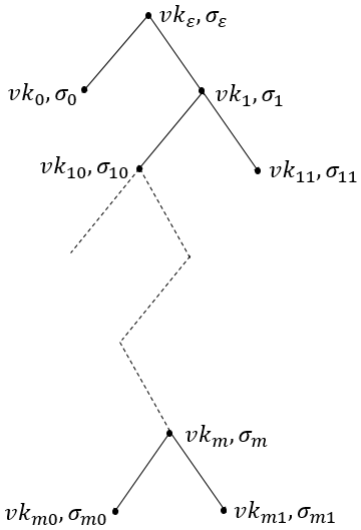


Figure 1: A signature on message m includes verification keys vk_{x0}, vk_{x1} and signatures σ_x on the path from the root to m with intermediate nodes x. The keys (sk_x, vk_x) are computed by $Gen(1^n; F_s(x))$ and signatures σ_x by $sign_{sk_x}(vk_{x0}, vk_{x1})$.

The Construction: In what follows $(Gen', Sign', Ver')$ is a one-time signature for arbitrary length messages and F is a PRF for arbitrary length inputs.

- $Gen(1^n)$: sample a PRF seed $s \in \{0,1\}^n$. Compute $(sk_\varepsilon, vk_\varepsilon) \leftarrow Gen'(1^n; F_s(\varepsilon))$. Save s as the secret key sk and publish vk_ε as the verification key vk.
- $Sign_{sk}(m)$: For every prefix x of m (including the empty one ε):
  - compute $sk_x, vk_x \leftarrow Gen'(1^n; F_s(x))$.

– compute $\sigma_x = Sign_{sk_x}(vk_{x0}, vk_{x1})$.

Output $\{vk_{x0}, vk_{x1}, \sigma_x\}_x$.

- $Ver_{vk}(m, \sigma)$: verify all the signatures from the root to the node m.

Note that indeed this scheme is stateless, and the size of the keys is fixed independently of the size of signed messages. The length of the signatures themselves does scale with the length of messages (this can also be dealt with using extra hashing.)

Claim 1.1. *The above scheme is secure assuming the one-time scheme is secure.*

Proof Sketch. First, note that the generator algorithm Gen' makes can be described as an efficient oracle-aided algorithm $Gen'^{F_s(\cdot)}$ that makes black-box calls to the sample PRF. Accordingly, it suffices to prove that the scheme is secure when Gen' is given access to a truly random function R, instead of a PRF. Indeed, any adversary that breaks the scheme when Gen' has access to a PRF, will break the scheme also when Gen' has access to a random function with the same probability up to a negligible difference. Otherwise, such an adversary could be used to efficiently distinguish a PRF oracle from a random function oracle (indeed, the entire signature game can be efficiently emulated).

Next, we show that when Gen' is given a random oracle, the scheme is secure. Let us assume a n.u. PPT adversary $A = \{A_n\}_n$ that breaks the multi-message scheme with probability $\varepsilon(n)$. We will use A to construct an adversary $A' = \{A'_n\}_n$ that breaks the one-time signature scheme with probability at least $\frac{\varepsilon(n)}{\ell(n)}$, where $\ell(n) = 2n \cdot q(n) + 1$ and $q(n)$ is an upper bound on the number of queries A makes. Note that ℓ is an upper bound on the number of keys that are relevant to A's queries.

Given a one-time verification key vk, A' will sample $i^* \leftarrow [\ell]$ and construct a list $\left(sk^i, vk^i\right)_{i\in[\ell]}$ as follows:

1. If $i \neq i^*$, $(sk_i, vk_i) \leftarrow Gen(1^n)$;
2. Otherwise $vk_i = vk$ (and sk_i will not be used).

We simulate $A(vk_\varepsilon := vk^1)$ by assigning signing and verification keys to the nodes in the tree in the following lazy fashion: When A asks for a signature on a message m:

1. For each prefix x of m whose corresponding node has not been assigned keys, assign the next pair of keys $(sk^i, vk^i), (sk^{i+1}, vk^{i+1})$ on the list as (sk_{x0}, vk_{x0}), (sk_{x1}, vk_{x1}) (we keep a counter so that each pair of keys is assigned to at most one node).
2. Compute $\sigma_x = Sign_{sk_x}(vk_{x0}, vk_{x1})$. For every $i \neq i^*$ the signature is computed using the secret keys sampled by A', whereas for i^*, the signature is computed using the (one-time) signature oracle.
3. Output $\{vk_{x0}, vk_{x1}, \sigma_x\}_x$.

Given A's answer m^*, if its signature $\sigma^* = \{vk^*_{x0}, vk^*_{x1}, \sigma^*_x\}_x$ is valid and m^* is a message which hasn't been signed before, then the following is true: There exists a prefix x of m^* where $(vk^*_{x0}, vk^*_{x1}) \neq (vk_{x0}, vk_{x1})$, or the latter pair has not been assigned. We take the shortest such prefix x. If $vk_{i^*} = vk_x$, A' outputs the signature σ^*_x on the message (vk^*_{x0}, vk^*_{x1}).

We first notice that the view of the emulated A is identical to its view in the actual experiment of the multi-message scheme. Hence, the probability that A will successfully forge a signature is $\varepsilon(n)$. In the case that A succeeds, if in addition $vk_{i^*} = vk_x$, then A' succeeds in forging a valid signature. Since i^* is sampled at random and independently of the sampled keys (and thus also of A's view), $vk_{i^*} = vk_x$ with probability $\frac{1}{\ell(n)}$. Overall, A'_n will succeed with probability $\frac{\varepsilon(n)}{\ell(n)}$. □

How to Get Rid of Collision Resistance. To reduce the underlying assumption from CRHs to OWFs, the crucial observation is that a weaker form of collision resistance is sufficient. In this weaker version, we only require that for any (adversarially chosen) input x it is hard to find a sibling x' such that $H_{hk}(x) = H_{hk}(x')$, when hk is chosen at random (after x).

Definition 1.2 (Targeted Collision-Resistant Hash (TCRH))**.** *A targeted collision-resistant hash function is given by a polynomial time algorithm H that given a key $hk \in \{0,1\}^n$ and input $x \in \{0,1\}^*$ outputs a hash $H_{hk}(x) \in \{0,1\}^n$. Furthermore, for any n.u. PPT $A = \{A_n\}_{n\in\mathbb{N}}$ and any sequence $\{x_n\}_{n\in\mathbb{N}}$ of polynomial-size strings, there exists a negligible μ such that for all $n \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{c|c} H_{hk}(x_n) = H_{hk}(x') & hk \leftarrow \{0,1\}^n \\ x_n \neq x' & x' \leftarrow A_n(hk, x_n) \end{array}\right] \leq \mu(n) \ .$$

Remark: a common definition in the literature allows A_n to sample x. In the non-uniform setting, this is equivalent to the above definition.

Theorem 1.3 ([NY89, Rom90])**.** *If there exist OWFs, then there exist TCRHs.*

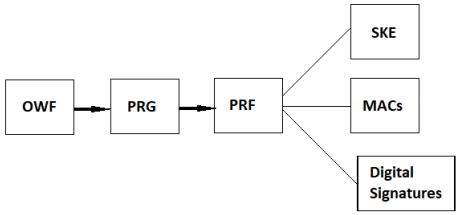
The proof of the above theorem is not trivial and is out of the scope of this course. Let us give some intuition though on why such hash functions are enough for signature schemes. For this, we need to go back to our length extension for one-time signatures. If we simply replace CRHs with TCRHs as is in the construction we've seen it won't work (try to think why). The observation is that in the one-time signature game the adversary first commits to some query message m, and only then gets a signature, which gives us the opportunity to sample a key for the TCRH. Concretely, the new signature would be

$$hk, Sign_{sk}(hk, H_{hk}(m))$$

(try to think why this works).

2 Privately Communicating without a Secret Key?

We've seen that starting from one-way functions, we can go a long way. We've constructed PRGs, PRFs, secret-key encryption, authentication and even digital signatures.

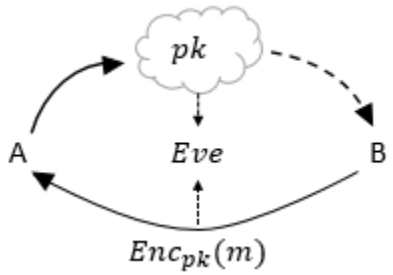


Today, we will move beyond into the world of public-key encryption. From the dawn of cryptography it was evident that *to communicate secretly, parties have to share a secret*, and this perception dominated for centuries. In the 70's, People had begun to challenge this perception, suggesting that secret communication may be achievable without the parties ever meeting! Curiously, these ideas emerged both publicly within the scientific community, and (a few years before that) behind closed doors within the British intelligence agency (you can read more about this fascinating story in Barak's notes).

One of the first to challenge the need for a secret key was Ralph Merkle, then an undergrad at Berkeley. He both conjectured that this is possible, and gave some evidence for this possibility (he demonstrated a public key encryption system where the attacker has to work quadratically, rather than super-polynomially, harder than the honest parties). The paper that officially started the age of public-key cryptography was published in '76 by Diffie and Hellman, and was titled "New Directions in Cryptography" [DH76]. It describes the concepts of public key-exchange, public-key encryption, and digital signatures, and suggested generic approaches and concrete constructions.

3 Public-Key Encryption

In the setting of public key encryption, Alice shares with the world (say, on her webpage) a *public encryption key pk* and keeps a corresponding secret key *sk*. Then any person, Bob, in hold of *pk* can encrypt messages to Alice. While she can decrypt, any eavesdropper Eve that doesn't know the secret key (but does know the public key) learns nothing.



Definition 3.1 (Public-Key Encryption (PKE) against Chosen Plaintext Attacks (CPA))**.** *A public-key encryption scheme consists of polynomial-time algorithms* (Gen, Enc, Dec) *with the following properties.*

- **Syntax:**
  - $Gen(1^n)$ *is a randomized algorithm that takes as input a security parameter* n *and outputs a secret key* sk *and a public encryption key* pk*.*
  - $Enc_{pk}(m)$ *is a randomized algorithm that takes as input the public key* pk *and message* $m \in \{0,1\}^*$*, and outputs a ciphertext* ct*.*
  - $Dec_{sk}(ct)$ *is a deterministic algorithm that takes as input the secret key* sk *and ciphertext* ct*, and outputs a decrypted message.*
- **Correctness:** *for any message* $m \in \{0,1\}^*$*,*

$$\Pr\left[Dec_{sk}(Enc_{pk}(m)) = m \mid (sk, pk) \leftarrow Gen(1^n)\right] = 1 \ .$$

- **Indistinguishability against Chosen Plaintext Attack:** *For any (interactive) n.u. PPT* $A = \{A_n\}$ *there is a negligible* μ *such that* A *wins the following CPA game with probability at most* $1/2+\mu(n)$*.*

  The CPA Game:
  - $Gen(1^n)$ *samples* sk, pk*.*
  - A_n *is given* pk*, and chooses two equal-length messages* m_0, m_1*.*
  - A_n *is then given an encryption* $Enc_{pk}(m_b)$ *for a random* $b \leftarrow \{0,1\}$*.*
  - A_n *wins if it guesses* b *correctly.*

Remarks:

- **Encryption Queries:** Typically, in a chosen plaintext attack, the adversary may also get oracle access to the encryption algorithm, allowing her to adaptively compute encryptions of messages of her choice and choose the challenge messages m_0, m_1 accordingly. In the case of public-key encryption, the above definition (without an explicit oracle) is equivalent (make sure you understand why).
- **Multi-Message Security:** The definition addresses a single challenge ciphertext. In the homework, you will show that it, in fact, implies multi-message security.

- **Public-Key Encryption vs Key Exchange:** A relaxation of the notion of public-key encryption is that of *key exchange*. Here two parties may run an *interactive protocol* at the end of which they agree on a secret key sk, such that an external observer that sees the transcript of the protocol cannot recover sk. It is not hard to show that PKE is equivalent to two-message key exchange. However, key exchange protocols with more than two rounds are not known to imply PKE.

Why Would PKE Exist? One has to ask what made Diffie, Hellman, and the other co-discoverers of public-key encryption believe that such a thing is even possible. It seems that one intuition they had is the following. Imagine that parties can not only send messages but also *magic boxes*. A sender party can lock a computer program Π in such a box so that the receiver can provide inputs $x$to the program and get the outputs $\Pi(x)$, but the box is otherwise opaque, and does not allow to see the contents of Π itself.

In this case, public key encryption can be done as follows. The public key pk can be a "publicly-verifiable puzzle", say $f(x)$ for a OWF f, and the secret key sk would be the solution x. Then to encrypt m, we will simply create a magic box and lock inside it the program $\Pi_{f(x),m}$ that returns m only when given the solution x. Another natural idea is to publish such a box as the public key and put inside the encryption algorithm of a secret key encryption algorithm (and say derandomize the encryption algorithm using a PRF).

Diffie and Hellman believed that it might be possible to digitally realize such a magic box, by simply *rewriting* Π in a way that will make it hard to read, e.g. by writing it in some low-level assembly-like language (or asking an undergrad to code it). Indeed, in general, reverse engineering is a notoriously hard problem. This idea is known today as *program obfuscation*. Diffie and Hellman didn't give any formal definitions or constructions, but rather used it as intuition to why PKE is conceivable. (In time, it had turned out that obfuscation is a fascinating concept beyond the context of PKE. It is also very challenging to define and construct. In fact, currently, there is still no satisfying solution to the problem. Hopefully, we will return to it toward the end of the course.)

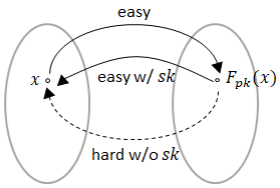
4 Basing PKE on Hard Problems

The question now is how to fulfill Diffie and Hellman's intuition based on reasonable computational assumptions. So far in the course we've been lucky enough to manage to construct everything from OWFs. It would be great if we could also prove:

Theorem 4.1 (In Our Dreams)**.** *If OWFs exist so do public-key encryption schemes.*

Unfortunately, we're very far from being able to prove such a theorem, and the common belief today is that public-key encryption requires considerably more structure than OWFs.

Trying to capture the structure required for PKE, Diffie and Hellman suggested the following generalization of one-way functions to trapdoor one-way functions. Here, a function is associated with a public key and a secret *trapdoor key*. Using the public key, anyone could compute the function forward, and using the secret key it can also be computed backwards (we'll assume for now that such functions are injective). However, for a random public key, and without the secret key, the function is one way.



Definition 4.2 (Trapdoor Function (TDF))**.** *A trapdoor function is given by polynomial time algorithms* (Gen, F, F^{-1}) *with the following properties:*

- **Syntax:**
  - $Gen(1^n)$ *is a probabilistic algorithm that given a security parameter* 1^n *outputs a secret key (also called the trapdoor) and a public key* (sk, pk)*.*
  - $F_{pk}(x)$ *is a deterministic algorithm that given a public key* pk *and input* $x \in \{0,1\}^n$*, outputs an image* y*.*
  - $F_{sk}^{-1}(y)$ *is a deterministic algorithm that given a secret key* sk *and* $y \in \{0,1\}^*$*, outputs* $x \in \{0,1\}^n$*.*
- **Correctness:** *for any* $x \in \{0,1\}^n$*,*

$$\Pr\left[F_{sk}^{-1}(F_{pk}(x)) = x \;\middle|\; (sk, pk) \leftarrow Gen(1^n)\right] = 1 \ .$$

- **One-Wayness:** *for any n.u. PPT* $A = \{A_n\}$*, there exists a negligible* μ *such that for all* $n \in \mathbb{N}$*,*

$$\Pr\left[A_n(pk, F_{pk}(x)) = x \;\middle|\; \begin{array}{r}(sk, pk) \leftarrow Gen(1^n) \\ x \leftarrow \{0,1\}^n\end{array}\right] \leq \mu(n) \ .$$

On the surface, TDFs seem like a simpler object than PKE. In particular, they only guarantee that random messages x are hard to fully recover, but they certainly don't guarantee to fully hide any input x. In fact, for any two messages x, x' you can always easily tell $F_{pk}(x)$ and $F_{pk}(x')$, as F_{pk} is deterministic. Historically, this is still how people imagined using TDFs for several years, until the work of Goldwasser and Micali [GM82] who introduced *probabilistic encryption* and the requirement that encryption leaks nothing about plaintexts, which later became the gold standard.[1]

Formally, however, public-key encryption is not known to be stronger than TDFs — they are not known to imply TDFs (think why a PKE scheme isn't already a TDF by itself). In contrast, the other direction is true.

Theorem 4.3. *TDFs imply PKE.*

Proof Sketch. Let (Gen, F, F^{-1}) be a TDF. We define a PKE (Gen, E, D) for one-bit messages.

- The Gen algorithms are the same (as suggested by our notation).
- $E_{pk}(m)$ samples $x, r \leftarrow \{0,1\}^n$, and outputs

$$F_{pk}(x), r, \langle x, r\rangle \oplus m \ .$$

- $D_{sk}(y, r, b)$ outputs $\langle F_{sk}^{-1}(y), r\rangle \oplus b$.

[1] This requirement, which we have so far formalized as indistinguishability, also has an alternative formulation known as semantic security, which we might mention later on.

The correctness of the scheme is obvious. As for security, by the Goldreich-Levin theorem, we know that

$$pk, F_{pk}(x), r, \langle x, r\rangle \approx_c pk, F_{pk}(x), r, u \ ,$$

where $pk \leftarrow Gen(1^n), x, r \leftarrow \{0,1\}^n, u \leftarrow \{0,1\}$.

The above already implies that

$$pk, E_{pk}(0) \approx_c pk, E_{pk}(1) \ .$$

In the homework, you will prove that in the case of PKE, the above is equivalent to CPA security:

Claim 4.4. *A bitwise PKE is CPA secure if and only if* $pk, E_{pk}(0) \approx_c pk, E_{pk}(1)$.

□

Diffie and Hellman defined trapdoor functions and demonstrated how to use them for encryption (although, in a deterministic fashion. This is before Goldwasser-Micali and Goldreich-Levin). However, they did not provide a candidate for such functions (they did provide a direct construction of PKE, which we'll touch in a bit).

RSA. The first trapdoor function that (publicly) emerged to the world in 1977 was suggested by Rivest, Shamir, and Adleman [RSA78], which became to be known as the RSA function. The trapdoor function, or actually permutation (TDP), is based on a number-theoretic problem related to factoring. We'll briefly present it, just to get a sense of how it looks, without getting too much into details.

- Let $p < q$ be two primes of magnitude $\Theta(2^n)$, and let $N = pq$.
- Let $\mathbb{Z}_N^*$ be the multiplicative group modulo N (these are all the numbers in $[N]$ that are coprime to N, and multiplication is done modulo N).
- The order of this group is $\phi(N) = (p-1)(q-1)$.
- A public key pk for the function consists of N and an integer $e \in \mathbb{Z}_{\phi(N)}^*$. The secret key sk consists of an integer $d \in \mathbb{Z}_{\phi(N)}^*$ such that $ed = 1 \mod \phi(N)$ (such an integer can be computed using the GCD algorithm, by those who know p and q).
- The function is defined (and in fact is a permutation) over $x \in \mathbb{Z}_N^*$ as follows

  $$F_{N,e}(x) = x^e \mod N \ .$$

  (This operation can be computed efficiently in time $\text{polylog}(N) = \text{poly}(n)$.)
- The inversion for $y \in \mathbb{Z}_N^*$ is given by

  $$F_d^{-1}(y) = y^d \mod N \ .$$

Assumption 4.5 (RSA Assumption, somewhat informally stated)**.** *For randomly chosen* p, q *and* e, $F_{N,e}(x)$ *is hard to invert for a random* x.

How hard is RSA? Today the best algorithms to solve this problem are those that factor N, but no formal reduction to factoring is known. (As we've already mentioned in previous lectures, factoring itself is not as hard as one may hope. It can be solved in subexponential time and also quantumly.)

Other TDFs and PKEs. In 1979, Rabin [Rab79] suggested another TDP based on the hardness of factoring N's of a special form (called Blum integers). For a very long time (roughly 30 years) these were the only known TDFs.[2] The most common way of constructing PKE schemes is directly based on different (or perhaps, as we'll later mention, not so different) computational problems. In fact, such a construction was already given by Diffie and Hellman. The construction was actually described as a two-message key exchange protocol. Here we'll present a slight variant of it as a PKE (which reflects the equivalence between the notions).

Diffie-Hellman PKE. The scheme comes from the *discrete log* problem in groups, which is perhaps one of the oldest and most investigated problems in computational number theory. Specifically, consider the multiplicative group $\mathbb{Z}_p^*$ for a prime $p \sim 2^n$. The discrete logarithm problem in this group is given $g, g^x \in \mathbb{Z}_p^*$, find x. The scheme is defined as follows (in what follows all group operations are $\mod p$).

- A public key pk consists of p, g, g^x where g is chosen at random from $\mathbb{Z}_p^*$, and $x \in \mathbb{Z}_{p-1}$ is a random exponent. The secret key sk consists x.
- The encryption algorithm is defined for bit messages as follows

$$E_{p,g,g^x}(m; y, r) = g^y, \langle g^{xy}, r\rangle \oplus m, r \ ,$$

  where $y \leftarrow \mathbb{Z}_{p-1}$ and $r \leftarrow \{0,1\}^n$, are randomness used by the encryption algorithm.
- Decryption for $h \in \mathbb{Z}_p^*$, $b \in \{0,1\}$, and $r \in \{0,1\}^n$ is given by

$$D_x(h, b, r) = \langle h^x, r\rangle \oplus b \ .$$

It is not hard to see that if one can solve the discrete log problem in $\mathbb{Z}_p$, then the scheme is completely insecure, but this is not enough. The security of the scheme is based on the following assumption.

Assumption 4.6 (Computational Diffie-Hellman Assumption (CDH), somewhat informally stated)**.** *For randomly chosen p and g, x, y, given p, g, g^x, g^y, it is hard to find g^{xy}.*

Claim 4.7. *Assuming CDH, the Diffie-Hellman PKE described above is CPA secure.*

Proof Sketch. By Claim 4.4, it is sufficient to show that

$$pk, E_{pk}(0) \approx_c pk, E_{pk}(1) \ .$$

For this it suffices to show that,

$$p, g, g^x, g^y, \langle g^{xy}, r\rangle, r \approx_c p, g, g^x, g^y, b, r$$

where b is a uniformly independent bit (make sure you understand why this is enough). Indeed, assume towards contradiction, there exists an efficient distinguisher A for the above distributions with advantage $\epsilon(n) = n^{-O(1)}$. Then, we can use A to break the CDH assumption. Specifically, we can construct from A an efficient predictor P such that

$$\Pr_{p,g,x,y,r}[P(p, g, g^x, g^y, r) = \langle g^{xy}, r\rangle] \geq \frac{1}{2} + \Omega(\varepsilon) \ .$$

The predictor P samples a random bit β and checks whether $A(p, g, g^x, g^y, \beta, r) = 1$. If so it returns β, and otherwise $1-\beta$. A standard calculation shows that this predictor works assuming $\Pr[A(p, g, g^x, g^y, b, r) = 1] - \Pr[A(p, g, g^x, g^y, b, r) = 1] \geq \varepsilon$. (If it is the other way around, we flip P's output.)

By averaging with probability $\Omega(\varepsilon)$ over p, g, x, y, it holds that

$$\Pr_r[P(p, g, g^x, g^y, r) = \langle g^{xy}, r\rangle] \geq \frac{1}{2} + \Omega(\varepsilon) \ .$$

In this case, we can apply to Goldreich-Levin algorithm to recover g^{xy}. □

[2] Today, we know of some additional ones, but these two constructions are still the only known permutations based on assumptions that are considered standard.

A famous variant of this assumption is the Decision DH (DDH), that says g^{xy} is not only hard to compute, but also pseudorandom. It naturally leads to a simpler variant of the above PKE (without the GL hardcore bit) known as ElGamal encryption. The DDH assumption, in fact, does not hold in $\mathbb{Z}_p^*$, but it is conjectured to hold in appropriate groups.

A Scheme of a Different Nature Based on LPN. We now briefly describe yet another system by Aleknovich [Ale03] based on the learning parity with noise problem (which we've already encountered previously in the course).

- A public key pk consists of $\hat{A} = (A, As + e)$ where $A \leftarrow \{0,1\}^{m \times n}$ (where $m \approx 2n$), $s \leftarrow \{0,1\}^n$ and $e \in \{0,1\}^m$ is a noise vector, where each coordinate is one with probability δ. The secret key sk consists of e.
- The encryption algorithm is defined for bit messages as follows:
  - To encrypt a one, output a random vector $u \leftarrow \{0,1\}^m$.
  - To encrypt a zero, sample a random y in the kernel of $\hat{A}^T$ and an error vector e' (distributed as e) and output $y + e'$.
- To decrypt $c \in \{0,1\}^m$, output $\langle c, e \rangle$.

When choosing the noise rate δ to be small enough (roughly $1/\sqrt{n}$), it can be shown that zero encryptions are decrypted correctly with probability close to 1 and one encryption with probability close to $1/2$. Correctness is then amplified by taking say n copies of the scheme (and using the fact that PKE remains secure under such repetition). Security is shown based on the LPN assumption, saying that

$$A, As + e \approx_c A, u \ .$$

On Basing PKE on Different Assumptions: All and all hard problems sufficient for PKE seem far more rare than those needed for secret-key encryption. In fact, we can essentially classify them into two groups of similar problems: 1) number/group theoretic problems (solved quantumly) and 2) lattice/decoding problems (believed to be quantum resistant). Coming up with essentially different PKE schemes is a major open problem.

References

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307, 2003.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377, 1982.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.

[Rab79] Michael O. Rabin. *Digitalized signatures and public-key functions as intractable as factorization.* MIT Laboratory for Computer Science, 1979. URL: `http://ncstrl.mit.edu/Dienst/UI/2.0/Describe/ncstrl.mit_lcs/MIT/LCS/TR-212`. Note: Technical Report 212.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394, 1990.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.