1 Public-key encryption

We now turn to the study of public-key encryption. In private-key encryption, we had assumed that Alice and Bob share a random secret key $K \in \{0,1\}^k$ which is completely unknown to Eve. One practical issue with this setup is that the parties need to get together and agree on this key in a secure environment. While this may be possible in physical settings, it is not clear how this key agreement can be achieved on the internet.

Public-key encryption offers a possible solution to this problem. In public-key encryption, Alice has a pair of keys, a secret key SK and a public key PK. The secret key is known only to Alice, while the public key is known to everyone, including the adversary. If Bob wants to send a message to Alice, he encrypts it with her public key. However, although the public key is sufficient for encryption, it is useless for decryption: The only way to recover the message (or extract any information about it) requires knowledge of the secret key.

We achieved private-key cryptography by starting with an information-theoretic requirement – perfect security – that could be in principle achieved but requires unreasonably large resources in practice. We then relaxed the security requirement in order to obtain a more reasonable implementation. In contrast, public-key cryptography is not even possible to do in a perfect security setting, no matter what the size of the keys.

Defining public-key encryption rigorously is a bit trickier than the private-key case. These complications arise mainly from the problem of choosing the keys. In the private-key setting, we could assume – essentially without loss of generality – that the key is a random string in $\{0,1\}^k$. In the public-key setting, the keys PK and SK will also be random, but they must also be related in some special way in order to make encryption and decryption complementary to each other. So we will assume that the pair (PK, SK) is generated as the output of a special randomized *key generation algorithm Gen*.

It also happens that in many (but not all) constructions of public-key encryption, plaintexts and ciphertexts do not come from sets like $\{0,1\}^k$, but rather from some other sets D, usually groups. The set D is usually random itself; it depends on the outcome of the key generation algorithm.

Definition 1. A public-key encryption scheme is a triple of algorithms (Gen, Enc, Dec) where

- Gen is a randomized algorithm that takes no input and produces a pair of keys (SK, PK) (and a domain D).
- Enc is a randomized algorithm that takes a public key PK and a message $M \in D$ and outputs a ciphertext $Enc(PK, M)$.
- Dec is an algorithm that takes a secret key SK and a ciphertext C and produces a plaintext $Dec(SK, C)$.
- For every message $M \in D$,
$$Dec(SK, Enc(PK, M)) = M$$

where (SK, PK) is the output of Gen, with probability one over the randomness of Gen and Enc.

Let us reflect a bit on how we would apply this definition in a practical scenario. If Bob wants to send a message to Alice, then he would run the encryption, and Alice would run the decryption. But who runs the key generation algorithm Gen? One possibility is for Alice to run her own key generation, store the secret key in a secure place, and share the public key with Bob (and anyone else who may wish to send her a message). On the internet, maybe she could do this by posting her public key on a web page, for example. But if her channel to Bob is not authenticated – for example if Eve impersonates Alice – then Bob may be operating under the wrong impression that he is encrypting using Alice's public key, while he may in fact be using someone else's public key. Various solutions to this problem have been proposed. We will not discuss them and we will assume that the public key is indeed known to everyone.

We now define message-indistinguishability for public-key encryption:

Definition 2. A public-key encryption scheme (Gen, Enc, Dec) is (s, ε) message indistinguishable if for every circuit A of size s and every pair of circuits M_0, M_1 of size s,

$$|\Pr[A(PK, Enc(PK, M_0(PK))) = 1] - \Pr[A(PK, Enc(PK, M_1(PK))) = 1]| \leq \varepsilon$$

where the randomness is taken over Gen, Enc, and A.

Notice a difference from the private-key setting: since the adversary is given access to the public key, the challenge messages M_0 and M_1 can also depend on the public key. This is why we model them as circuits.

By giving the adversary access to the public key, we give him the ability to compute arbitrary encryptions of his choice. So in the public-key setting, CPA security reduces to message indistinguishability. An immediate consequence is that any message indistinguishable scheme must be probabilistic.

Also notice that a "perfectly secure" variant of this definition (i.e., with s arbitrary) can never be achieved, regardless of the key size: Whenever $M_0 \neq M_1$, for any PK the possible encryptions $Enc(PK, M_0)$ and $Enc(PK, M_1)$ must be disjoint, so a computationally unbounded adversary can always recover the message from its encryption.

How do we construct secure public-key encryption? Ideally, we would like to proceed as in private-key cryptography: We identify a simple primitive, like a pseudorandom generator, that is necessary and sufficient for the construction of public-key encryption and give a scheme assuming the security of the primitive. Unfortunately no such primitive is known in the public-key setting. (In particular, there is evidence that public-key encryption requires more than the existence of pseudorandom generators.)

We will shortly define an object called a trapdoor permutation, a primitive that *can* be used to construct public-key encryption. However, not all public-key encryption schemes are based on trapdoor permutations. But to give some intuition about how public-key encryption schemes look like, we begin by describing a "direct" construction: the El Gamal encryption scheme.

2 The El-Gamal encryption scheme

The El Gamal encryption scheme is based on a computational hardness assumption called the *Decisional Diffie Hellman (DDH) assumption*. Let G be a cyclic group of size q with generator g. We say the (s, ε) DDH assumption holds for (G, g) if for every circuit of size s,

$$|\Pr_{x,y\sim Z_q}[A(g^x, g^y, g^{xy}) = 1] - \Pr_{x,y,z\sim Z_q}[A(g^x, g^y, g^z) = 1]| \leq \varepsilon.$$

The DDH assumption is related to the hardness of taking discrete logarithms. If the discrete logarithm problem in G was easy – namely, given h we could find an x so that $g^x = h$ – then the DDH assumption would not hold: On input (a, b, c) we take the discrete logs of a and b and output 1 if G raised to their product equals c. This happens with probablity 1 on the left, but with probability at most $1/q$ on the right.

However, the hardness of computing discrete logarithms is in general not sufficient for (G, g) to satisfy the DDH assumption. For example, when p is a random prime number, and g is a generator for $\mathbb{Z}_p^*$, it is believed that the discrete logarithm problem is usually hard in $\mathbb{Z}_p^*$, but it is known that $\mathbb{Z}_p^*$ does *not* satisfy the DDG assumption.

One way to obtain a group for which the DDH assumption may be true is like this. Choose a number p where both p and $q = (p-1)/2$ are prime numbers. Let h be a generator for $\mathbb{Z}_p^*$ and Q be the subgroup of $\mathbb{Z}_p^*$ generated by $g = h^2$. Then Q is a cyclic group of order q and it is plausible that the DDH assumption holds with high probability for such groups.[1]

We now describe the El Gamal encryption scheme. Next time we will argue that this scheme is message indistinguishable provided that the DDH assumption holds:

> **Key Generation:** Choose a random prime p, so that $(p-1)/2$ is also prime, $p \leq 2^k$, and a random $h \in \mathbb{Z}_p^*$. Let $g = h^2$. Choose a random $x \leq (p-1)/2$. Output the pair $SK = (p, g, x)$, $PK = (p, g, g^x)$.
>
> **Encryption:** $Enc((p, g, g^x), M) = (g^r, g^{xr} \cdot M)$ where $M \in Q$ and r is chosen at random from $\mathbb{Z}_q$.
>
> **Decryption:** $Dec((p, g, x), (C, C')) = C'/C^x$.

It is easy to see that this scheme is correct. To argue that it is secure, we need to show that for every pair of small circuits M_0, M_1, the distributions $(p, g, g^x, g^r, g^{xr} \cdot M_0)$ and $(p, g, g^x, g^r, g^{xr} \cdot M_1)$ are indistinguishable, where M_0 and M_1 may depend on the public key (p, g, g^x). To do this, we will compare both distributions with the distribution (p, g, g^x, g^r, g^z) and argue that if they are distinguishable, we can break the DDH assumption.

Claim 3. *Suppose the (s, ε) DDH holds with probability $1-\varepsilon$ for the cyclic subgroup of $\mathbb{Z}_p^*$ generated by g. Then El Gamal encryption is $(s/2, 4\varepsilon)$ message indistinguishable.*

Proof. Suppose not. Then there is are circuits A, M_0, M_1 of size $s/2$ so that

$$|\Pr[A(p, g, g^x, g^r, g^{xr} \cdot M_0) = 1] - \Pr[A(p, g, g^x, g^r, g^{xr} \cdot M_1) = 1]| > 4\varepsilon.$$

[1] I don't think it is known that there are infinitely many primes p where $(p-1)/2$ is also a prime, but it seems plausible that there are in fact many such primes.

(To keep notation simple, we omit the input $PK = (p, g, g^x)$ of M_0 and M_1 from this description.) By a hybrid argument, for at least one $b \in \{0, 1\}$ we must have:

$$|\Pr[A(p, g, g^x, g^r, g^{xr} \cdot M_b) = 1] - \Pr[A(p, g, g^x, g^r, g^z) = 1]| > 2\varepsilon.$$

Then

$$\Pr_{p,g}\big[|\Pr_{x,r}[A(p, g, g^x, g^r, g^{xr} \cdot M_b) = 1] - \Pr_{x,r,z}[A(p, g, g^x, g^r, g^z) = 1]| > \varepsilon\big] > \varepsilon.$$

Fix a pair (p, g) for which the expression inside is greater than ε. For this pair (p, g), we give a circuit B that (s, ε) breaks the DDH assumption: On input (a, b, c), output $A(a, b, c \cdot M_b(p, g, a))$. Then B has size s and

$$\Pr_{x,r}[B(g^x, g^r, g^{xr}) = 1] = \Pr[A(p, g, g^x, g^r, g^{xr} \cdot M_b) = 1]$$

while

$$\Pr_{x,r,z}[B(g^x, g^r, g^z) = 1] = \Pr[A(p, g, g^x, g^r, g^z \cdot M_b) = 1] = \Pr[A(p, g, g^x, g^r, g^z) = 1]$$

because g^z is uniformly random in Q and independent of x and r. It follows that the (s, ε) DDH assumption fails to hold for more than ε of the pairs (p, g). □

There is one implementation issue regarding El Gamal encryption: The messages reside in a group Q, which is a subgroup of $\mathbb{Z}_p^*$. How do we represent bit strings as elements of Q? In our example, Q consists of the *quadratic residues* in $\mathbb{Z}_p^*$ (namely those elements that are perfect squares), and so a message coming from the set $\{0, \ldots, q-1\}$ can be represented by its square modulo p. To decode the message from its square, we need to take square roots modulo p.

How do we do this? Given $y \in \mathbb{Z}_p^*$, we need to find x so that $x^2 = y$. Since y is a quadratic residue, it must be that $y = g^{2t}$ for some t, so we need to solve the equation $x^2 = g^{2t}$. This equation has two solutions given by $x = \pm g^t$. It looks like in order to find x, we need to know the discrete log of y. But there is a trick that avoids taking discrete logs: Since $g^{p-1} = g$, we have that $g^{2t} = g^{(p+1)t}$, and so the square roots of g^{2t} are the values $\pm g^{(p+1)t/2} = \pm y^{(p+1)/4}$. So taking the square root of y is the same as raisng y to the power $(p+1)/4$ (as long as p and $q = (p-1)/2$ are both primes).

3 Trapdoor permutations

Let's go back to generic public-key encryption and consider intuitively the requirements that a public-key scheme should satisfy. For this, it will help to think of decryption as the "inverse" of encryption:

- For every public key, encryption should be easy to compute.
- If we know the public key but do not know the secret key, encryption should be hard to invert.
- If we know the secret key, encryption becomes easy to invert.

Notice that these requirements subsume the requirements of a one-way function: Easy to compute, but hard to invert. However there is an additional requirement: Encryption should become easy to

invert in the presence of some extra information, the secret key. We call this a *trapdoor*: it allows us to magically go back where we came from.

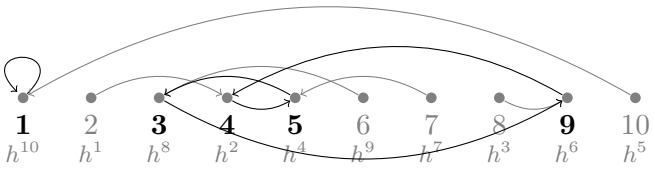
This suggests that the core of public-key encryption is the existence of "trapdoor functions". Unfortunately it turns out that this intuition is not quite correct.[2] However, if the function happens to be a permutation, then we can indeed obtain public-key encryption.

Definition 4. A family of functions f_{PK} is a (s, ε) *trapdoor permutation family* with key generation algorithm Gen and trapdoor inversion algorithm Inv if

- For every possible output (SK, PK) of Gen, the function f_{PK} is a permutation.
- For every circuit A of size at most s, $\Pr[A(PK, f_{PK}(x)) = x] \leq \varepsilon$ where the probability is taken over the choice of $(SK, PK) \sim Gen$ and x chosen at random from the domain of f_{PK}.
- For every pair $(SK, PK) \sim Gen$ and every x in the domain of f_{PK}, $Inv(SK, f_{PK}(x)) = x$.

Trapdoor permutations are somewhat difficult to come by. We now describe a conjectured construction.

Quadratic residuosity To describe this construction we need a bit of algebra. As before, let p be a prime number so that $(p-1)/2$ is also a prime, h a generator of $\mathbb{Z}_p^*$, and $Q_p \subseteq \mathbb{Z}_p^*$ the subgroup of quadratic residues generated by h^2. To get some intuition about Q_p, let's look at $\mathbb{Z}_{11}^*$:



The arrows indicate the graph of the map $x \to x^2$. The numbers in bold are the quadratic residues. Notice that each quadratic residue has two square roots: One inside Q_p (the black incoming edge) and one outside Q_p (the gray incoming edge). In particular, the map $x \to x^2$ is a permutation of Q_p. However, this permutation is easy to invert: As we said before, the square roots of y are $\pm y^{(p+1)/4}$.

We can generalize these observations to every group $\mathbb{Z}_p^*$ where $(p-1)/2$ is also a prime number. If h is a generator for Z_p^*, the set of quadratic residues is $Q_p = \{h^2, h^4, \ldots, h^{p-1} = 1\}$. All these elements have distinct squares: For if $(h^{2a})^2 = (h^{2b})^2$, then $p-1$ must divide $4a - 4b$, so $(p-1)/2$ must divide $a - b$, and so $a = b$. Therefore the map $f(x) = x^2$ is a permutation of Q_p, and its inverse is the map $y \to y^{(p+1)/4}$.

Now suppose $n = pq$ is a product of two distinct prime numbers, and let us assume that both $(p-1)/2$ and $(q-1)/2$ are primes also. The group $\mathbb{Z}_n^*$ consists of those numbers that have a multiplicative inverse modulo n. These are all the numbers that are not multiples of p or multiples of q (or both). The elements of $\mathbb{Z}_n^*$ can be identified with pairs of elements, one in $\mathbb{Z}_p^*$, the other

[2] One reason for this is that a "trapdoor function" is a deterministic object, while encryption may (and should) be randomized.

one in $\mathbb{Z}_q^*$ by the representation map $u \to (u \bmod p, u \bmod q)$. This is a one-to-one map from $\mathbb{Z}_n^*$ to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.

What do the quadratic residues Q_n of $\mathbb{Z}_n^*$ look like? First, notice that if $y \in Q_p$ and $z \in Q_q$, then the pair (y, z) is in Q_n, and it has four square roots represented by the pairs

$$(-y^{(p+1)/4}, -z^{(q+1)/4}) \quad (-y^{(p+1)/4}, z^{(q+1)/4}) \quad (y^{(p+1)/4}, -z^{(q+1)/4}) \quad (y^{(p+1)/4}, z^{(q+1)/4})$$

Since $Q_p \times Q_q$ has a quarter of the size of $\mathbb{Z}_n^*$, and each pair $(y, z) \in Q_p \times Q_q$ has exactly four square roots, it follows that Q_n contains exactly those elements that are represented by pairs in $Q_p \times Q_q$. Moreover, exactly one of these four square roots comes from $Q_p \times Q_q$, so the map $x \to x^2$ is a permutation of Q_n. This is our candidate one-way permutation:

> Let p and q be prime numbers where $(p-1)/2$ and $(q-1)/2$ are also prime and Q_n be the subgroup of quadratic residues of $\mathbb{Z}_n^*$. Then the map $f(x) = x^2$ is a permutation over Q_n.

Is the permutation f one-way? Suppose there is a circuit A that inverts f with probability ε. Now choose a random $x \in \mathbb{Z}_n^*$ and look at the output of $A(x^2)$. Since x^2 is a random element of $\mathbb{Z}_n^*$, with probability ε, $A(x^2)$ will output a square root u of x^2. But didn't we already know a square root? Yes, but there are four, and u could be different from the square root we already know. In fact, conditioned on x^2 (represented by (y, z)), x is equally likely to take any of the four values

$$(-y^{(p+1)/4}, -z^{(q+1)/4}) \quad (-y^{(p+1)/4}, z^{(q+1)/4}) \quad (y^{(p+1)/4}, -z^{(q+1)/4}) \quad (y^{(p+1)/4}, z^{(q+1)/4}).$$

To be specific, let us assume that $u = A(x^2)$ is represented by $(y^{(p+1)/4}, z^{(q+1)/4})$. Then with probability at least $1/4$, x is represented by $(y^{(p+1)/4}, -z^{(q+1)/4})$, and therefore $x + u$ is represented by $(2y^{(p+1)/4}, 0)$, which is a multiple of q. Then the GCD of n and $x + u$ must equal q.

To summarize, we just showed that if A inverts f on an ε-fraction of its outputs, then the following circuit produces one of the factors of n with probability at least $\varepsilon/4$:

> On input $n = pq$:
> Choose $x \in \mathbb{Z}_n^*$ at random.
> Run $A(x^2)$ to obtain u. Output the GCD of n and $x + u$.

By repeating this algorithm $O(1/\varepsilon)$ times, we can increase the chances of finding a factor of n from $\varepsilon/4$ to a constant. So if the permutation f is easy to invert, then it is almost equally easy to factor the number n.

It is possible that factoring numbers like n could be computationally intractable:

Hardness of factoring assumption (for parameters (s, ε))**:** Let $p \neq q$ be random primes between 1 and 2^k so that $(p-1)/2$ and $(q-1)/2$ are also prime. Then for every circuit A of size s,

$$\Pr_{A,p,q}[A(pq) = p \text{ or } A(pq) = q] \leq \varepsilon.$$

The best known factoring algorithm (for numbers of any form) appears to run in time about $2^{\sqrt[3]{k}}$. Using a quantum algorithm, it is possible to factor in time polynomial in k. The quantum

algorithm, however, does not have a scalable implementation, and it is a matter of debate whether it is a credible threat to the hardness of factoring assumption.

The hardness of inverting f relies heavily on the assumption that the factors of n are difficult to find. If the factors of n are known, it turns out that f is easy to invert. We already know that the square roots of $u \in Q_n$ are the numbers represented by

$$(-y^{(p+1)/4}, -z^{(q+1)/4}) \quad (-y^{(p+1)/4}, z^{(q+1)/4}) \quad (y^{(p+1)/4}, -z^{(q+1)/4}) \quad (y^{(p+1)/4}, z^{(q+1)/4}).$$

To find a square root of u, all that remains is to figure out which numbers in $\mathbb{Z}_n^*$ are represented by these pairs (and which one of them resides in Q_n, which we can figure out by attempting another square root). If we know p and q, there is a formula for inverting such representations:

$$\text{if } (y, z) = (u \bmod p, u \bmod q), \text{ then } u = zp \cdot (p^{-1} \bmod q) + yq \cdot (q^{-1} \bmod p).$$

To summarize, the following algorithm inverts the map $f(x) = x^2$ given access to p and q:

$Inv((p, q), u)$:
Let $(y, z) = (u \bmod p, u \bmod q)$.
Compute the representations u_1, u_2, u_3, u_4 of $(\pm y^{(p+1)/4}, \pm z^{(q+1)/4})$.
If $u_1^2 \neq u$, return `error` ($u \notin Q_n$).
Return u_i, where i is the unique index for which $Inv((p, q), u_i)$ does not return `error`.

In conclusion, it is plausible that the function f is a trapdoor permutation: It is hard to invert assuming the hardness of factoring assumption, but easy to invert if we are given the factorizaton of n. Let *Gen* be an algortihm that chooses random primes $p, q \leq 2^k, p \neq q$ where $(p-1)/2$ and $(q-1)/2$ are also prime, and outputs the pair $SK = (p, q)$, $PK = n$. We just argued that

Theorem 5. *Assume the (s, ε) hardness of factoring assumption holds. Then $\{f \colon Q_n \to Q_n\}$ is an $(s/k^{O(1)}, \Omega(\varepsilon))$ trapdoor permutation family with key generation algorithm Gen and trapdoor inversion algorithm Inv.*

The analysis remains the same if the assumptions that $(p-1)/2$ and $(q-1)/2$ are prime are weakened to the requirement that they are merely odd numbers.

4 Public-key encryption from trapdoor permutations

In the private-key setting, our encryption schemes were naturally based on pseudorandom generators. A pseudorandom generator, in turn, can be constructed from a one-way permutation and a hard-core bit. Once one bit of pseudorandomness is obtained, the length of the output can be increased by carefully constructed iterative applications of the pseudorandom generator.

An analogous sequence of transformations can be used in the public-key setting to obtain public-key encryption from trapdoor permutations. To illustrate this, we begin by designing a one-way encryption scheme from a trapdoor permutation and its hardcore bit.

Definition 6. Let $\{f_{PK} \colon D_{PK} \to D_{PK}\}$ be a family of trapdoor permutations with key generation *Gen* and trapdoor inversion *Inv*. A function family $\{h_{PK} \colon D_{PK} \to \{0,1\}\}$ is a (s, ε) *hardcore bit family* for $\{f_{PK}\}$ if for every circuit P of size s,

$$\Pr_{PK,x}[P(PK, f_{PK}(x)) = h_{PK}(x)] \leq \frac{1}{2} + \varepsilon.$$

Just as in the case of one-way permutations, if h_{PK} is a hardcore bit for f_{PK}, the pair $(f_{PK}(x), h_{PK}(x))$ is difficult to distinguish from the pair (y, b), where y is a random element in the domain of f_{PK} and b is a random bit. This suggests that $h_{PK}(x)$ can be used to hide a one-bit message $M \in \{0,1\}$ via the following encryption scheme:[3]

$$\begin{aligned} Enc(PK, M) &= (f_{PK}(R), h_{PK}(R) + M) \quad \text{where } R \sim D_{PK} \text{ is random} \\ Dec(SK, (Y, B)) &= B + h_{PK}(Inv(SK, Y)). \end{aligned}$$

Claim 7. *Assume $\{h_{PK}\}$ is an (s, ε) hardcode bit family for $\{f_{PK}\}$. Then (Gen, Enc, Dec) is $(s/2 - O(1), 2\varepsilon)$ message indistinguishable.*

Proof. If not, then there are circuits A, M_0, M_1 of size $s/2 - O(1)$ so that

$$\begin{aligned} |\Pr[A(PK, f_{PK}(R), h_{PK}(R) + M_0(PK))) = 1] \\ - \Pr[A(PK, f_{PK}(R), h_{PK}(R) + M_1(PK))) = 1]| > \varepsilon \end{aligned}$$

so there is some $b \in \{0,1\}$ for which

$$|\Pr[A(PK, f_{PK}(R), h_{PK}(R) + M_b(PK))) = 1] - \Pr[A(PK, f_{PK}(R), B) = 1]| > \varepsilon$$

where $B \sim \{0,1\}$ is a random bit. Let D be a circuit that on input (PK, y, c) returns $A(PK, y, c + M_b(PK))$. Then D has size $s - O(1)$ and

$$|\Pr[D(PK, f_{PK}(R), h_{PK}(R)) = 1] - \Pr[D(PK, f_{PK}(R), B) = 1]| > \varepsilon.$$

By the same argument that we used in the last lecture, using D we can construct a predictor P of size s for which

$$\Pr[P(PK, f_{PK}(x)) = h_{PK}(x)] > \frac{1}{2} + \varepsilon$$

violating our assumption. □

One annoying thing about this scheme is that it can only encrypt single bits. To do better, we need to obtain multiple hardcore bits, which we can do using the same approach that allowed us to extend the output of pseudorandom generators. To encrypt messages in $\{0,1\}^m$, we use the following encryption:

$$Enc(PK, M) = (f_{PK}^{(m)}(R), h_{PK}(R) + M_1, h_{PK}(f_{PK}(R)) + M_2, \ldots, h_{PK}(f_{PK}^{(m-1)}(R)) + M_m)$$

where $f^{(i)}(x) = f(f(\ldots f(x)\ldots))$ a total of i times. The proof of security is very similar to the proof of Theorem 9 from Lecture 2.

Under the hardness of factoring assumption, there are several known hardcore bits for the function $f(n, x) = x^2 \bmod n$. For example, the least significant bit of x is known to be a hardcore bit.

[3] We will assume that the public key is implicitly contained in the secret key.