# Public Key Encryption

KG

November 25, 2016

## Contents



# 1 Introduction

In this note, we consider the following problem. Alice wants to send a message to Bob via some channel. Eve has access to the channel and she may eavesdrop on anything sent over the channel. Alice does not want Eve to know the content of her message to Bob.

The obvious solution is for Alice and Bob to first run the Diffie-Hellman protocol to establish a shared secret. Then Alice can use the shared secret to encrypt her message using symmetric cryptography.

For some channels, such as mail, this is very inconvenient, since each message may take a long time to arrive, and even longer before it is read and acted upon.

Of course, Alice and Bob could establish a shared secret and then simply use that secret from then on. But Alice may need to talk to many people, not just Bob. Sharing secrets with all of them and then managing the shared secrets is inconvenient.

Alice wants to be able to send a single encrypted message to Bob or one of her other correspondents. She does not want to manage shared secrets with every correspondent, but she may be willing to manage public information, just as she is already managing names, phone numbers and adresses.

The answer to Alice's problem is public key encryption, and the basic idea is explained and defined in Section 2. Section 3 describes a public key encryption scheme based on the Diffie-Hellman protocol, and Section 4 describes a public key encryption scheme based on arithmetic in certain rings.

This text is intended for a reader that is familiar with mathematical language, basic number theory, basic algebra (groups, rings, fields and linear algebra) and elementary computer science (algorithms), as well as the Diffie-Hellman protocol.

This text is informal, in particular with respect to computational complexity. Every informal claim in this text can be made precise, but the technical details are out of scope for this note.

This text uses colour to indicate who is supposed to know what. When discussing cryptography, red denotes secret information that is only known by its owner, Alice or Bob. Green denotes information that Alice and Bob want to protect, typically messages. Blue denotes information that the adversary Eve will see. Information that is assumed to be known by both Alice and Bob (as well as Eve) is not coloured.

We also use colour for theorems about computation, where blue denotes information that an algorithm gets as input and can use directly, while red denotes information that exists, but has to be computed somehow before it can be used directly. Information that is considered fixed (such as the specific group in use, group order, generator, etc.) and that the algorithm may depend on is not coloured.

# 2 Public Key Encryption

Alice, Bob and a number of other people want to be able to send confidential messages to each other. For various reasons, using the Diffie-Hellman protocol to establish a shared secret every time they want to send messages is not possible or practical. Furthermore, Alice does not want to manage a long-term secret for each correspondent. She is willing to manage public information for each correspondent.

In this situation, what is needed is public key encryption.

**Definition 1.** A *public key encryption* scheme consists of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:

- The *key generation* algorithm $\mathcal{K}$ takes no input and outputs an *encryption key* $ek$ and a *decryption key* $dk$. To each encryption key $ek$ there is an associated message set $\mathcal{M}_{ek}$.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key $ek$ and a message $m \in \mathcal{M}_{ek}$ and outputs a ciphertext $c$.

- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key $dk$ and a ciphertext $c$ and outputs either a message $m$ or the special symbol $\perp$ indicating decryption failure.

We require that for any key pair $(ek, dk)$ output by $\mathcal{K}$ and any message $m \in \mathcal{M}_{ek}$

$$\mathcal{D}(dk, \mathcal{E}(ek, m)) = m.$$

Just as for symmetric encryption, the users of a system require confidentiality and some sense of integrity. However, since the encryption key is known anyone can create a ciphertext, so the informal notion of integrity does not work for public key encryption. Instead, we have a notion of non-malleability where it should be hard to modify a ciphertext in a predictable way.

**Definition (informal) 2.** A public key encryption scheme provides *confidentiality* if it is hard to learn anything at all about the decryption of a ciphertext from the ciphertext itself, possibly except the length of the decryption.

**Definition (informal) 3.** A public key encryption scheme is *non-malleable* if it is hard to create a new ciphertext based on a given ciphertext such that the decryption of the new ciphertext is not $\perp$, but predictably related to the given ciphertext.

# 3 Schemes Based on Diffie-Hellman

We shall develop a public key encryption scheme based on the Diffie-Hellman protocol. The initial situation is that Alice, Carol and Bob will use the Diffe-Hellman protocol to establish a shared secret, and then send the message encrypted using a symmetric encryption scheme.

Let $G$ be a finite cyclic group of order $n$ and let $g$ be a generator. Let $(G, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ be a symmetric cryptosystem. Note that $G$ is the key set for the symmetric cryptosystem.

When Alice wants to send a message $m_A \in \mathcal{P}$ to Bob, she uses Diffie-Hellman to establish a shared secret, encrypts her message using the symmetric cryptosystem and sends the ciphertext to Bob. Bob decrypts the ciphertext.

1. Alice chooses a number $r_A$ uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. She computes $x_A = g^{r_A}$ and sends $x_A$ to Bob.

2. Bob receives $x_A$ from Alice. He chooses a number $b_A$ uniformly at random from the set $\{0, 1, \ldots, n-1\}$, computes $y_A = g^{b_A}$ and $z_A = x_A^{b_A}$.

3. Alice receives $y_A$ from Bob. Alice computes $z_A = y_A^{r_A}$, encrypts the message as $w \leftarrow \mathcal{E}_s(z_A, m_A)$, and sends $w$ to Bob.

The situation where both Alice and Carol send messages to Bob is illustrated in Figure 1.

But a variation on this topic is possible. Before anything happens, Bob executes part of the Diffie-Hellman protocol. He samples a random number $b$ and computes $y = g^b$.

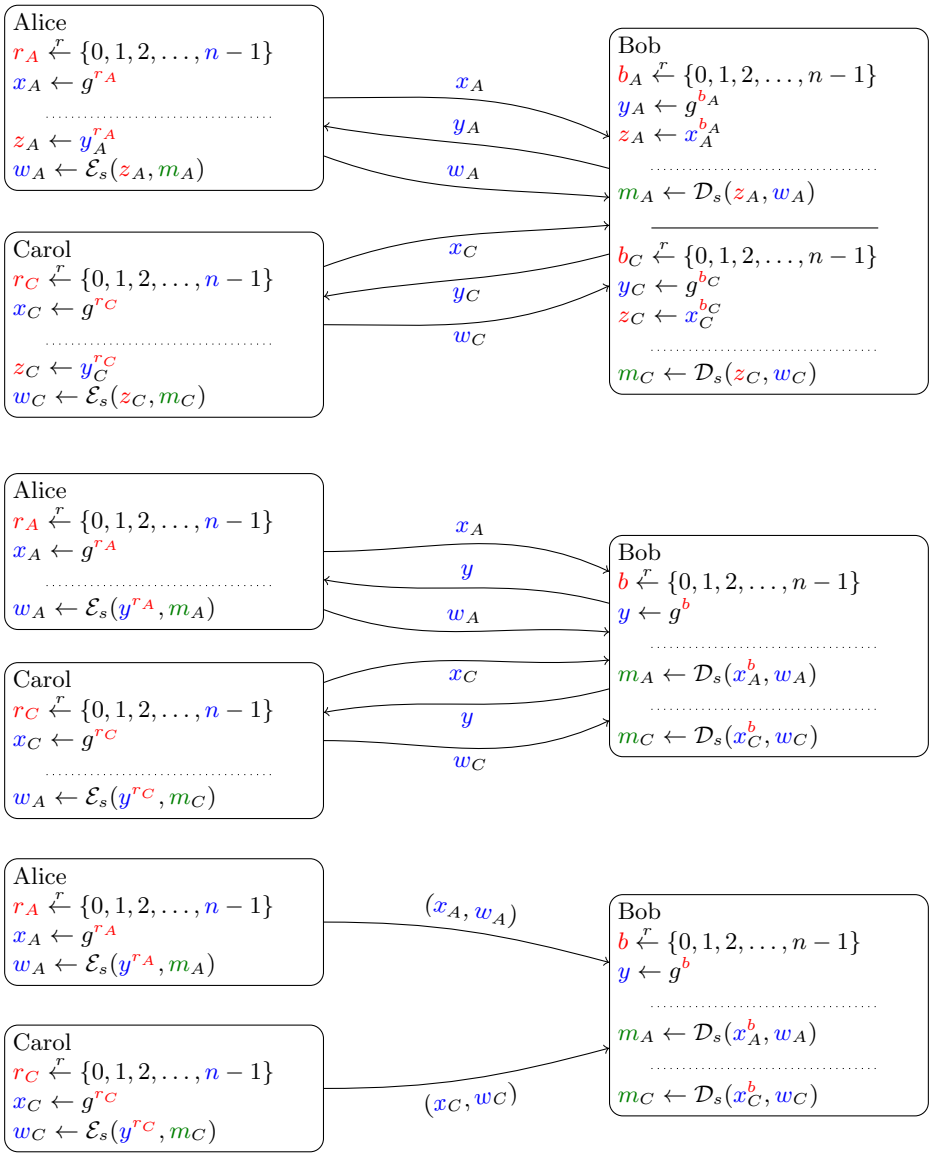


Figure 1: From Diffie-Hellman to public key encryption scheme. (top) Alice and Carol use Diffie-Hellman to establish shared secrets with Bob and send him encrypted messages. (middle) Bob uses a single random number for all the Diffie-Hellman protocol runs. (bottom) Bob publishes his Diffie-Hellman message. Alice and Carol complete the Diffie-Hellman protocol to establish a shared secret and send Bob encrypted messages.

Whenever someone contacts him, he immediately responds with $y$. When he receives the symmetric ciphertext, he computes the shared secret and decryptes the ciphertext. This variation is illustrated in the middle part of Figure 1.

It is possible to prove that Bob does not lose any security by doing this. The proof is out of scope for this note.

Obviously, repeatedly sending the same value $y$ is wasteful. Bob therefore announces publicly what he is doing and publishes the value $y$. When Alice and Carol want to send a message to Bob, they already know $y$. Therefore, they do not have to wait to receive it. Instead, they can immediately compute the shared secret and encrypt the message. This final situation is shown in the bottom of Figure 1.

What has happened is that we have turned the Diffie-Hellman protocol combined with a symmetric cryptosystem into a public key encryption scheme.

The public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is based on a finite cyclic group $G$ of order $n$ with generator $g$ and a symmetric cryptosystem $(G, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$.

- The *key generation* algorithm $\mathcal{K}$ samples a number $b$ uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. It computes $y = g^b$ and outputs $ek = y$ and $dk = b$. The message set associated to $ek$ is $\mathcal{P}$.

- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key $y$ and a message $m \in \mathcal{P}$. It samples a number $r$ uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. Then it computes $x = g^r$ and $z = y^r$, and encrypts the message as $w = \mathcal{E}_s(z, m)$. It outputs the ciphertext $c = (x, w)$.

- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key $b$ and a ciphertext $c = (x, w)$. It computes $z = x^b$ and decrypts the message as $m = \mathcal{D}_s(z, w)$. If $\mathcal{D}_s$ outputs the special symbol $\perp$ indicating decryption failure, then $\mathcal{D}$ outputs $\perp$, otherwise it outputs $m$.

*Exercise* 1. The above is an informal description of a public key encryption scheme. Write down carefully what the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

## 3.1 ElGamal

The security of the above scheme depends both on the security of the Diffie-Hellman protocol and the security of the symmetric cryptosystem.

In order to illustrate certain attacks, it is convenient to consider a variant of the above public key encryption scheme that uses an extremely simple symmetric cryptosystem, namely a variant of the Shift cipher given by $(G, G, G, \mathcal{E}_s, \mathcal{D}_s)$, where $\mathcal{E}_s(k, m) = mk$ and $\mathcal{D}_s(k, w) = wk^{-1}$.

The resulting scheme is known as the *ElGamal* (or *textbook ElGamal*) public key encryption scheme.

- The *key generation* algorithm is as described above.

- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key $y$ and a message $m \in G$. It samples a number $r$ uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. Then it computes $x = g^r$ and $w = my^r$. It outputs the ciphertext $c = (x, w)$.
- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key $b$ and a ciphertext $c = (x, w)$. It computes $m = wx^{-b}$.

*Exercise* 2. The above is an informal description of a public key encryption scheme. Write down carefully what the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

**Information Leakage** For certain groups, the ElGamal public key cryptosystem may leak information about the message. This may or may not be problematic.

*Exercise* 3. Suppose $p$ is a prime such that $p - 1$ is divisible by a large prime and the discrete logarithm problem is hard in $\mathbb{F}_p^*$. Consider ElGamal based on $G = \mathbb{F}_p^*$. Let $y$ be the encryption key, and let $c = (x, w)$ be an encryption of $m$ under $y$. Show that

$$\left(\frac{m}{p}\right) = \begin{cases} \left(\frac{w}{p}\right) & \text{if } \left(\frac{x}{p}\right) \text{ or } \left(\frac{y}{p}\right) \text{ is 1; and} \\ -\left(\frac{w}{p}\right) & \text{otherwise.} \end{cases}$$

*Exercise* 4. Suppose $p$ is a prime such that the discrete logarithm problem is hard in $\mathbb{F}_p^*$. Suppose also that $p - 1$ is divisible by a (small) known prime $\ell$, and that $m \in \mathbb{F}_p^*$ has order $\ell$. Show that $m$ can be recovered from an encryption of $m$ using essentially a small multiple of $\sqrt{\ell}$ group operations.

**Malleability** When the attacker only sees the encryption key and one ciphertext, we say that we have a *chosen plaintext attack*. Quite often the attacker will be able to deduce some information about the decryption of other ciphertexts. We typically consider a situation where the adversary wants to learn something about the decryption of one or more ciphertexts, and may ask for the decryption of one or more other ciphertexts. This is known as a *chosen ciphertext attack*.

We begin by showing that ElGamal is *malleable*, in that even if you do not know the decryption of a ciphertext, it is still possible to create new ciphertexts that decrypt to the same or related messages.

*Exercise* 5. Suppose $c = (x, w)$ is an encryption of an unknown message $m$. Show how to create an encryption $c' = (x', w')$ of $m$ where $x' \neq x$. Also show how to create an encryption of $mm'$ for any $m' \in G$.

We can now use these properties of ElGamal to attack confidentiality under a chosen ciphertext attack.

*Exercise* 6. Suppose Alice sends Bob the ciphertext $c = (x, w)$ and Eve learns this ciphertext. Suppose further that Eve is capable of tricking Bob into decrypting one ciphertext that is different from $c$. Show how Eve can learn the decryption of $c$.

The above exercises show that ElGamal, and in particular ElGamal over $\mathbb{F}_p^*$, does not provide confidentiality. Neither does ElGamal provide non-malleability.

However, if the symmetric cryptosystem used provides confidentiality and integrity, it can be proven under reasonable assumptions that the public key encryption scheme from Exercise 1 is non-malleable and provides confidentiality.

# 4 RSA

In this section we shall develop the famous RSA public key encryption scheme. We begin with the Pohlig-Hellman exponentiation cipher, which can be interpreted as a public key encryption scheme, albeit an insecure one. The problem is that if the group order is known, it is easy to recover the decryption key from the encryption key.

The exponentiation cipher can be adapted to a different algebraic structure, where we are able to prove that recovering a decryption key is as hard as factoring certain integers. While this does not prove that the scheme is secure if it is hard to factor those integers, it turns out that factoring seems to be the best way to attack the cryptosystem. We show a number of flaws in this cryptosystem and discuss various ways of improving the system.

## 4.1 Preliminaries

The Pohlig-Hellman exponentiation cipher is based on a cyclic group $G$ of order $N$. The key is an integer $k$ relatively prime to $N$, and the two block cipher maps are

$$(k, x) \mapsto x^k \qquad \text{and} \qquad (k, y) \mapsto y^{k^{-1}},$$

where $k^{-1}$ is any inverse of $k$ modulo $N$.

Observe now that we can turn the Pohlig-Hellman cipher into a public key encryption scheme as follows. The key generation algorithm chooses an integer $e$ from the integers between 0 and $N$ that are invertible modulo $N$. It then finds some inverse $d$ of $e$ modulo the group order $N$. The encryption key $ek$ is $e$, while the decryption key $dk$ is $d$.

The encryption algorithm takes as input an encryption key $ek = e$ and a message $m \in G$ and outputs the ciphertext $c = m^e$.

The decryption algorithm takes as input a decryption key $dk = d$ and a ciphertext $c \in G$ out outputs the message $m = c^d$.

*Exercise* 7. The above is an informal description of a public key encryption scheme. Write down carefully what the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Unfortunately, this public key encryption scheme is trivially insecure. An adversary that sees the encryption key $e$ can easily compute an inverse of $e$ modulo the group order $N$, which is assumed known. (It may be impossible to compute the exact inverse found by the key generation algorithm, but any inverse can be used to compute the decryption map.)

However, if the group order was unknown, there would be no obvious way to compute a decryption key from the encryption key. The first thing we do is to note that Pohlig-Hellman works equally well in any finite group.

*Exercise* 8. Let $G$ be a finite group of order $N$, and let $e$ and $d$ be inverses modulo (a multiple of) $N$. Show that the maps $x \mapsto x^e$ and $y \mapsto y^d$ are inverse maps.

The key generation algorithm needs to compute inverses modulo the group order, which usually requires that the key generation algorithm knows the group order. This means that we cannot fix a single group. Instead, the key generation algorithm must choose a group in such a way that it knows the group order. Then it must include in the encryption key a description of the group such that the group order is hard to compute from that description.

Before we continue, we observe that for the Pohlig-Hellman cipher, $ed \equiv 1 \pmod{N}$, which means that $ed - 1$ is a multiple of $N$. Likewise, Exercise 8 says that if we know a multiple of the group order, we can find something that is effectively a decryption key.

## 4.2 The RSA Cryptosystem

Before we define the famous RSA cryptosystem, we extend the result of Exercise 8 to a specific *ring*.

*Exercise* 9. Let $n$ be the product of two large primes $p$ and $q$, and let $e$ and $d$ be inverses modulo (a multiple of) $\operatorname{lcm}(p-1, q-1)$. Show that the maps $x \mapsto x^e$ and $y \mapsto y^d$ on $\mathbb{Z}_n$ are inverses.

Hint: Prove that $p$ divides the difference $x^{ed} - x$ for any integer $x$. Repeat for $q$. Then conclude that the product divides the difference.

The *textbook RSA* public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ works as follows.

- The *key generation* algorithm $\mathcal{K}$ chooses two large primes $p$ and $q$. It computes $n = pq$, chooses $e$ and finds $d$ such that $ed \equiv 1 \pmod{\operatorname{lcm}(p-1, q-1)}$. Finally it outputs $ek = (n, e)$ and $dk = (n, d)$. The message set associated to $ek$ is $\{0, 1, 2, \ldots, n-1\}$.
- The *encryption algorithm* $\mathcal{E}$ takes as input an encryption key $(n, e)$ and a message $m \in \{0, 1, \ldots, n-1\}$. It computes $c = m^e \bmod n$ and outputs the ciphertext $c$.
- The *decryption algorithm* $\mathcal{D}$ takes as input a decryption key $(n, d)$ and a ciphertext $c$. It computes $m = c^d \bmod n$ and outputs the message $m$.

Note that the encryption exponent $e$ may be very small, even 3.

*Exercise* 10. The above is an informal description of a public key encryption scheme. Write down carefully (except for how to choose $p$, $q$ and $e$) what the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Hint: Use Exercise 9.

As we saw in Section 4.1, a public key encryption scheme is obviously useless if it is easy to deduce the decryption key from the encryption key. We shall now consider this problem for the RSA cryptosystem.

Let $p$ and $q$ be distinct, large primes, and let $n = pq$. Consider the group $\mathbb{Z}_n^*$, which is isomorphic to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$. It is clear that if we know $p$ and $q$, we can find $N = (p-1)(q-1)$. Conversely, if we know $n$ and $N$, then we can easily recover the factorization.

**Proposition 1.** *Let $n$ be a product of two distinct primes $p$ and $q$, and let $N$ be the order of $\mathbb{Z}_n^*$. Then $p$ and $q$ are zeros of the polynomial $f(X) = X^2 + (N - n - 1)X + n$.*

*Proof.* First note that $N = n - (p + q) + 1$, so

$$p + q = n + 1 - N.$$

Now consider the polynomial $f(X) = (X - p)(X - q) = X^2 - (p + q)X + n$. □

Note that if we know $N$, then we know the polynomial's coefficients, and if we know the coefficients, we can easily compute the zeros of the polynomial using the usual formula for the zeros of a quadratic polynomial.

If we only know $n$ and a multiple of $l$ of $N$, we cannot use the above result, but we can still recover the factorization of $n$.

*Exercise* 11. Let $n$ be a product of two distinct primes $p$ and $q$. Let $x$ and $y$ be integers such that

$$x \equiv y \pmod{p} \qquad \text{and} \qquad x \not\equiv y \pmod{q}.$$

Show that $\gcd(x - y, n) = p$.

**Lemma 2.** *Let $n$ be a product of two distinct large primes $p$ and $q$, and let $k$ be an odd multiple of $\operatorname{lcm}(p - 1, q - 1)/2$. Then for at least half of all integers $z$ between 0 and $n$ that are relatively prime to $n$,*

$$z^k \equiv \pm 1 \pmod{p} \qquad \text{and} \qquad z^k \equiv \mp 1 \pmod{q}. \tag{1}$$

*Proof.* Suppose first that neither $p - 1$ nor $q - 1$ divide $k$. Then $(p - 1)/2$ and $(q - 1)/2$ both divide $k$ and

$$\frac{k}{(p-1)/2} \qquad \text{and} \qquad \frac{k}{(q-1)/2}$$

are both odd, so the Legendre symbol tells us that the equations

$$x^k \equiv -1 \pmod{p} \qquad \text{and} \qquad y^k \equiv -1 \pmod{q}$$

hold for half of all integers $x \in \{1, 2, \ldots, p-1\}$ and half of all integers $y \in \{1, 2, \ldots, q-1\}$.

Otherwise, suppose without loss of generality that $q - 1$ divides $k$, while $p - 1$ does not divide $k$. Again, the Legendre symbol tells us that

$$x^k \equiv -1 \pmod{p} \qquad \text{and} \qquad y^k \equiv 1 \pmod{q}$$

for half of all integers $x \in \{1, 2, \ldots, p - 1\}$ and any integer $y \in \{1, 2, \ldots, q - 1\}$.

In either case, the Chinese remainder theorem says that (1) holds for half of all integers $z$ between 0 and $n$ that are relatively prime to $n$. □

**Proposition 3.** *Let $n$ be a product of two distinct primes $p$ and $q$. For any $\kappa > 0$, given a multiple $l$ of $N$, we can compute $p$ and $q$ with probability $1 - 2^{-\kappa}$ using at most $\lceil 4\kappa \log_2 l \rceil$ arithmetic operations and $\kappa \log_2 l$ gcd computations.*

*Proof.* Write $l = 2^t s$, $p-1 = 2^{t_p} s_p$ and $q-1 = 2^{t_q} s_q$ with $s$, $s_p$ and $s_q$ all odd. We may assume that $t_p \geq t_q$. It is then clear that $2^{t_p - 1} s$ is an odd multiple of $\operatorname{lcm}(p-1, q-1)/2$.

Lemma 2 then says that for half of all $z$ between 0 and $n$ that are relatively prime to $n$,

$$\gcd(z^{2^{t_p - 1} s} + 1, n) = p \text{ or } q.$$

For each value $z$ chosen uniformly at random from $\{x \mid 0 < x < n, \gcd(x, n) = 1\}$ the probability that the above gcd computation does not produce a proper factor of $n$ is $1/2$. The probability that a proper factor of $n$ has not appeared after $\kappa$ independently sampled $z$ values is $2^{-\kappa}$.

Obviously, we do not know $t_p$. But for each value of $z$ chosen, we can simply try all possible values for $t_p$. Since we know that $t_p \leq t < \log_2 l$, this requires at most $\log_2 l$ gcd computations per $z$ value.

We can compute the greatest common divisor by computing $z^{2^i s}$ modulo $n$ before computing the gcd. We can compute this using the standard recursive exponentiation algorithm using $4 \log_2 l$ arithmetic operations. By computing first $z^{2^0 s}$ first, we can compute $z^{2^1 s}, \ldots, z^{2^{t-1} s}$ successively, thereby computing all $t$ values modulo $n$ using just $4 \log_2 l$ arithmetic operations. Doing this for at most $\kappa$ distinct $z$ values then requires at most $4\kappa \log_2 l$ arithmetic operations. □

*Exercise* 12. The proof of Proposition 3 essentially describes an algorithm for factoring a product of two large primes $p$ and $q$ given a multiple $l$ of $\operatorname{lcm}(p-1, q-1)$. Write out this algorithm carefully and restate Proposition 3 as a statement about the algorithm's time complexity (in terms of arithmetic operations and gcd computations, ignoring any other form of computation involved).

Proposition 3 says that if anyone knows a multiple of $N$, then they can factor. It follows that if we believe that it is difficult to factor integers like $n$, then it is also hard to deduce the group order of $\mathbb{Z}_n^*$ or any multiple of it, and therefore it is hard to deduce the RSA decryption key from the RSA encryption key.

## 4.3 Attacks

We have seen that as long as the RSA modulus $n$ chosen by the key generation algorithm is hard to factor, the above public key encryption scheme is not obviously useless. In fact, it turns out that it is useful.

The best strategy for recovering a completely unknown $m$ from $c = m^e \bmod n$ — computing *$e$th roots* — seems to be to factor $n$. However, as the security definitions in Section 2 make clear, the adversary does not need to recover a completely unknown $m$ to break the system. In this section, we shall consider a few attacks on the public key encryption scheme from Exercise 10 that work essentially without computing $e$th roots.

**Deterministic Encryption** One fundamental problem with the public key encryption scheme is that it is not randomized. The ciphertext depends only on the message.

*Exercise* 13. Show that $\mathcal{D}(dk, c) = m$ if and only if $\mathcal{E}(ek, m) = c$.

*Exercise* 14. Let $S$ be a fixed and known set of messages, and let $c$ be an encryption of some message from $S$. Show that we can decide what the decryption of $c$ is using at most $|S|$ encryptions.

**Information Leakage** Part of the message encrypted will leak, which may or may not be a problem.

*Exercise* 15. Show that both $e$ and $d$ will be odd. Show that

$$\left(\frac{m}{n}\right) = \left(\frac{c}{n}\right).$$

**Malleability** Just like ElGamal, RSA is *malleable* and this can be used in a *chosen ciphertext attack*.

*Exercise* 16. Suppose $c$ is an encryption of an unknown message $m$. Show how to create an encryption $c'$ of $mm'$ for any $m'$ in the message space.

*Exercise* 17. Suppose Alice sends Bob the ciphertext $c$ and Eve learns this ciphertext. Suppose further that Eve is capable of tricking Bob into decrypting one ciphertext that is different from $c$. Show how Eve can learn the decryption of $c$.

**Short Messages** Just like we developed a public key encryption scheme based on Diffie-Hellman and a symmetric encryption scheme, we can combine RSA and a symmetric cryptosystem $(\mathcal{K}_s, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ to form a new public key encryption scheme.

The obvious idea is to consider the keys of the symmetric cryptosystem as integers and do as follows:

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The *encryption* algorithm takes an encryption key $ek = (n, e)$ and a message $m \in \mathcal{P}$ as input. It chooses a random key $k$ from $\mathcal{K}_s$, computes $x \leftarrow k^e \bmod n$ and $w \leftarrow \mathcal{E}_s(k, m)$. Finally, it outputs $c = (x, w)$.
- The *decryption* algorithm takes a decryption key $dk = (n, d)$ and a ciphertext $c = (x, w)$ as input. It computes $k \leftarrow x^d \bmod n$ and $m \leftarrow \mathcal{D}_s(k, w)$. If $k \notin \mathcal{K}_s$ or decryption of $w$ fails, it outputs $\perp$. Otherwise it outputs $m$.

One problem is that symmetric keys are typically small compared to the RSA modulus.

*Exercise* 18. Suppose $n \approx 2^{2000}$, $e = 3$ and $\mathcal{K}_s = \{0, 1, 2, \ldots, 2^{256} - 1\}$. Given an encryption $c$ of an unknown key $k$, show how you can easily recover $k$.

**Small Integer Roots of Polynomial Equations** A first solution to this problem is to add a large, fixed integer such as $2^{1999}$ to the key before raising it to the $e$th power, and subtracting it after raising the RSA ciphertext to the $d$th power. That is, $x \equiv (2^{1999}+k)^e \pmod{n}$. Unfortunately, for small $e$ it is easy to find small integer roots of modular univariate equations such as $(2^{1999}+X)^e - x \equiv 0 \pmod{n}$, and $k < 2^{256}$ is a small integer root of this equation.

A better solution is to add a random multiple of a large, fixed integer. That is, $x \equiv (r2^{256}+k)^e \pmod{n}$. We recover $k$ by computing the remainder when divided by $2^{256}$ after raising $x$ to the $d$th power.

While this seems to work, there are some worries. First, even if it is hard to compute $e$th roots in $\mathbb{Z}_n$, it may not be hard to compute parts of the root, e.g. the key $k$. Second, it may be possible to modify $x$ in such a way that $k$ does not change, only $r$. Since the decryption algorithm discards $r$, it will accept the modified ciphertext as valid, which may be a problem for certain applications.

## 4.4 Secure Variants

There is a simple, robust solution that uses a *random-looking* function to derive the key from a random number, which is raised to the $e$th power. Suppose $h$ is a function from $\{0,1,\ldots,n-1\}$ to $\mathcal{K}_s$. We get the following cryptosystem.

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The *encryption* algorithm takes an encryption key $ek = (n,e)$ and a message $m \in \mathcal{P}$ as input. It chooses $r$ uniformly at random from $\{0,1,\ldots,n-1\}$, computes $k \leftarrow h(r)$, $x \leftarrow r^e \bmod n$ and $w \leftarrow \mathcal{E}_s(k,m)$. Finally, it outputs $c = (x,w)$.
- The *decryption* algorithm takes a decryption key $dk = (n,d)$ and a ciphertext $c = (x,w)$ as input. It computes $r \leftarrow x^d \bmod n$, $k \leftarrow h(r)$ and $m \leftarrow \mathcal{D}_s(k,w)$. If decryption of $w$ fails, it outputs $\perp$. Otherwise, it outputs $m$.

Unless you know all of $r$, you know very little about the key $k$, since $h$ is random-looking. Furthermore, any change in $x$ will lead to a change in $r$, which will lead to an unpredictable change in $k$ because $h$ is random-looking. If $k$ has changed unpredictably, it will be hard to modify $w$ so that it is a valid ciphertext under the modified $k$, so it will be hard to get the decryption algorithm to say anything but $\perp$.

*Exercise* 19. The above is an informal description of a public key encryption scheme. Write down carefully what the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K},\mathcal{E},\mathcal{D})$ is a public key encryption scheme.

*Exercise* 20. Suppose you want to send the same message $m$ to $k$ recipients with public keys $(n_1,e_1),(n_2,e_2),\ldots,(n_l,e_l)$. One simple approach is to use the same $r$ and compute $x_i \leftarrow r^{e_i} \bmod n_i$, $i = 1,2,\ldots,k$, $k \leftarrow h(r)$ and $w \leftarrow \mathcal{E}_s(k,m)$. You send $(x_i,w)$ to the $i$th recipient.

Suppose $e_1 = e_2 = \cdots = e_l \leq l$. Show how you can easily recover $r$.

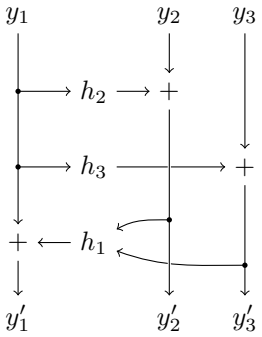


Figure 2: The permutation $\pi$ can be used for RSA encryption.

We note that Exercise 20 does not show an attack on the public key encryption scheme from Exercise 19. Instead, it illustrates one way to misuse a cryptosystem and thereby introduce weaknesses. In this case, reusing the randomness $r$ for more than one encryption introduced the weakness. In general, it is implicitly assumed that randomness used by key generation and encryption algorithms is never reused and does not leak out of the algorithm. If those assumptions are violated, weaknesses may result as shown by the exercise.

In situations with multiple recipients of the same message, other schemes may be more convenient. One solution is based on a Feistel-like permutation. Suppose we have three groups $G_1$, $G_2$ and $G_3$ such that $G_1 \times G_2 \times G_3$ can be considered a subset of $\{0, 1, \ldots, n-1\}$. Suppose also that we have three random-looking functions $h_1 : G_2 \times G_3 \to G_1$, $h_2 : G_1 \to G_2$ and $h_3 : G_1 \to G_3$. Then we can construct two permutations on $G_1 \times G_2 \times G_3$ as

$$\pi_1(y_1, y_2, y_3) = (y_1, y_2 + h_2(y_1), y_3 + h_3(y_1)) \text{ and}$$
$$\pi_2(y'_1, y'_2, y'_3) = (y'_1 + h_1(y'_2, y'_3), y'_2, y'_3).$$

The inverses of these two permutations are obvious. Our cryptosystem will use the composition $\pi = \pi_2 \circ \pi_1$ shown in Figure 2.

Suppose $\mathcal{K}_s \subseteq G_2$. Our cryptosystem works as follows.

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The *encryption algorithm* takes an encryption key $ek = (n, e)$ and a message $m \in \mathcal{P}$ as input. It chooses $r$ uniformly at random from $G_1$ and $k$ uniformly at random from $\mathcal{K}_s$, computes $x \leftarrow \pi(r, k, 0)^e \bmod n$ and $w \leftarrow \mathcal{E}_s(k, m)$. Finally, it outputs $c = (x, w)$.
- The *decryption algorithm* takes a decryption key $dk = (n, d)$ and a ciphertext $d$ as input. It computes $(r, k, y_3) \leftarrow \pi^{-1}(x^d \bmod n)$ and $m \leftarrow \mathcal{D}_s(k, w)$. If $k \notin \mathcal{K}_s$ or $y_3 \neq 0$ or the decryption of $w$ failed, it outputs $\perp$. Otherwise it outputs $m$.

*Exercise* 21. The above is an informal description of a public key encryption scheme. Write down carefully what the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Suppose we know $x$, that $x^d \bmod n = (y_1', y_2', y_3')$ and that $\pi^{-1}(y_1', y_2', y_3') = (r, k, 0)$.

Since $h_2(r)$ is added to the key to get $y_2'$, it is impossible to recover $k$ without knowing $r$. But $h_1$ is used to hide $r$, so it is impossible to recover all of $r$ without knowing both $y_1'$, $y_2'$ and $y_3'$.

Furthermore, any change in $y_1'$ will lead to a change in $r$, which will lead to an unpredictable change in $h_3(r)$. Any change in $y_2'$ or $y_3'$ will lead to an unpredictable change in $r$. If $r$ has changed, it is unlikely that $\pi^{-1}$ will result in 0 in the third coordinate, which means that the decryption algorithm will reject the ciphertext.

# 5 Factoring Integers

The cryptosystems described in Section 4.4 seem to be secure if it is hard to compute $e$th roots modulo the RSA modulus. It seems like the best method to compute $e$th roots modulo the RSA modulus $n$ is to factor $n$. To decide how hard it is to attack the cryptosystems in Section 4.4, we must therefore study how hard it is to factor RSA moduluses.

Throughout this section, unless otherwise stated we shall consider an integer $n$ that is the product of two large primes $p$ and $q$, with $q < p$. We do this to simplify the exposition, but we note that most of the factoring algorithms we consider will work for other composite numbers, and often work much better.

We shall usually estimate the work required by our algorithms in terms of arithmetic operations. By arithmetic operations, we mean additions, subtractions, multiplications or divisions of integers of about the same size as $n$. We ignore additions and subtractions of small constants, as well as multiplication and division by 2.

We begin with the simplest possible factoring algorithm, so-called trial division. While it works, it is extremely slow.

**Proposition 4.** *A factor of a composite integer $n$ can be found using at most $\sqrt{n}$ arithmetic operations.*

*Proof.* The composite $n$ must have a factor smaller than $\sqrt{n}$. We can divide $n$ by the integers $2, 3, 4, \ldots, \lfloor\sqrt{n}\rfloor$ in order. When the remainder is zero, we have found the smallest prime divisor of $n$. □

*Exercise* 22. The proof of Proposition 4 essentially describes an algorithm for finding a factor of a composite integer. Write out this algorithm carefully and restate Proposition 4 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

**Requirement 1.** Let $q$ be the smallest divisor of $n$. Then $q$ arithmetic operations should be an infeasible computation.

Having sufficiently large prime divisors is obviously easy to arrange.

*Exercise* 23. Suppose $n$ is a product of one or more prime powers, and that the biggest prime divisor is $p$. Show that the prime factorization of $n$ can be computed using at most $p + \log_2 n$ arithmetic operations.

## 5.1 Fermat Factoring

If $p$ and $q$ are numbers of roughly the same size, then $p$ and $q$ should be close to $\sqrt{n}$, but their average should be closer. Searching for the average close to $\sqrt{n}$ may make sense.

Let $t = (p+q)/2$ and let $s = (p-q)/2$. Then

$$t^2 - s^2 = (t+s)(t-s) = pq = n.$$

This means that we can decide if some integer $\tau$ equals $t$ by computing $\tau^2 - n$ and checking if it is an integer square.

**Lemma 5.** *Given an integer $k > 1$, we can compute the integer square root of $k$ if it exists using at most $3\log_2 k + 3$ arithmetic operations.*

*Proof.* We use binary search and construct a sequence of intervals $(l_1, r_1), (l_2, r_2), \ldots$ as follows.

Let $l_1 = 1$ and $r_1 = k$. Let $u_i = \lfloor (l_i + r_i)/2 \rfloor$. If $u_i^2 > k$, set $(l_{i+1}, r_{i+1}) = (l_i, u_i)$. If $u_i^2 < k$, set $(l_{i+1}, r_{i+1}) = (u_i, r_i)$. If $u_i^2 = k$, set $(l_{i+1}, r_{i+1}) = (u_i, u_i)$.

If $u_i^2 = k$ for some $i$, then $l_j = \sqrt{k} = r_j$ for all $j > i$. Otherwise, note that $1 < \sqrt{k} < k$. And if $l_i < \sqrt{k} < r_i$ and $u_i^2 \neq k$, then $l_{i+1} < \sqrt{k} < r_{i+1}$. It follows that the intervals satisfy $l_i \leq \sqrt{k} \leq r_i$ for all $i$. Furthermore, if $l_i \neq r_i$, then $l_i < \sqrt{k} < r_i$.

If $r_i - l_i > 1$, then $r_{i+1} - l_{i+1} \leq \lceil (r_i - l_i)/2 \rceil$, or alternatively $r_{i+1} - l_{i+1} \leq (r_i - l_i)/2 + 1/2$. It follows that

$$r_{i+1} - l_{i+1} \leq \frac{1}{2}(r_i - l_i) + \frac{1}{2} \leq \frac{1}{2}\left(\frac{1}{2}(r_{i-1} - l_{i-1}) + \frac{1}{2}\right) + \frac{1}{2}$$

$$\leq 2^{-i}(r_1 - l_1) + \sum_{j=1}^{i} 2^{-j} < 2^{-i}k + 1.$$

It follows $r_{i+1} - l_{i+1} \leq 1$ for $i > \log_2 k$.

We now have a simple algorithm for computing the integer square root, or concluding that it does not exist. It computes pairs $(l_i, r_i)$ until either $r_i = l_i$, in which case $r_i$ is the integer square root of $k$, or $r_i = l_i + 1$, in which case $k$ is not the square of any integer.

This algorithm will terminate after computing at most $\lceil \log_2 k \rceil$ pairs (after the first). Finally, given a pair $(l_i, r_i)$, computing $(l_{i+1}, r_{i+1})$ requires 3 arithmetic operations. The claim follows. □

*Exercise* 24. The proof of Lemma 5 essentially describes an algorithm for computing an integer square root, or proving that no such square root exists. Write out this algorithm carefully and restate Lemma 5 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

*Exercise* 25. Compute the square root of 16129 (by hand) using the algorithm from Exercise 24.

**Lemma 6.** *Let $n = pq$ where $p, q$ are large primes. Then $(p+q)/2 - \sqrt{n} \leq |p-q|/2$.*

*Proof.* The claim holds if $p = q$. We may therefore assume that $p > q$. With $t = (p+q)/2$ and $s = (p-q)/2$ as above, we have that

$$t - s = \sqrt{(t-s)^2} < \sqrt{(t-s)(t+s)} = \sqrt{n},$$

from which the claim follows. □

**Theorem 7** (Fermat factoring)**.** *Let $n$ be a product of two large primes $p, q$. Then a factor of $n$ can be found using at most $3|p-q|(2+\log_2 n)/2 + 1$ arithmetic operations.*

*Proof.* By Lemma 6, there is an $i < |p-q|/2$ and integer $\sigma$ such that

$$(\lceil \sqrt{n} \rceil + i)^2 - n = \sigma^2.$$

We can find the first integer square among the $|p-q|/2$ integers $(\lceil \sqrt{n} \rceil + i)^2 - n$ using at most $3 \log_2 n + 3 + 3$ arithmetic operations per integer, by Lemma 5.

Once we find $t = \lceil \sqrt{n} \rceil + i$ and the square root $s$, we find a proper factor $t - s$ of $n$ using a single arithmetic operation. The claim follows. □

*Exercise* 26. The proof of Theorem 7 essentially describes an algorithm for finding a factor of a composite integer. Write out this algorithm carefully and restate Theorem 7 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

*Exercise* 27. Factor 1683557 using the algorithm from the above proof.

*Exercise* 28. We can use a variant of the Sieve of Eratosthenes to find primes quickly. The idea is that we sieve over a small integer range to quickly exclude numbers divisible by small primes. The remaining numbers are then much more likely to be prime. This reduces the number of expensive primality tests we must do before we find a prime.

We can adapt this idea to RSA key generation by sieving over a range likely to contain two primes, which will be our $p$ and $q$. Explain why this is a bad idea.

We arrive at the following requirement.

**Requirement 2.** Let $n = pq$. Then $|p-q|$ arithmetic operations should be an infeasible computation.

*Exercise* 29. Suppose we choose two numbers $x$ and $y$ independently and uniformly at random from the range $\{2^{k-1}, 2^{k-1}+1, \ldots, 2^k - 1\}$. What is the expected value of $|x-y|$?

## 5.2 Pollard's $p-1$

We saw earlier that if we have a multiple of $\operatorname{lcm}(p-1, q-1)$, then we can factor $n$. However, it turns out that if we have a multiple of $p-1$ or $q-1$, we can usually also factor $n$.

**Proposition 8.** *Let $n$ be the product of two distinct large primes $p$ and $q$, and let $k$ be a multiple of $p-1$, but not a multiple of $q-1$. Then for at least half of all integers $z$ between 0 and $n$ that are relatively prime to $n$,*

$$z^k \equiv 1 \pmod{p} \qquad \text{and} \qquad z^k \not\equiv 1 \pmod{q}. \tag{2}$$

*Proof.* The first equation holds for all $z$ that are relatively prime to $n$.

Let $g$ be a generator for $\mathbb{F}_q^*$, and let

$$x = \frac{q-1}{\gcd(q-1, k)}.$$

Then $g^k$ has order $x$, which means that

$$z^k \equiv 1 \pmod{q}$$

for at most $1/x$ of all integers $z$ between 0 and $n$ that are relatively prime to $n$. The claim follows. □

Note that if we have $z$ and $k$ such that (2) holds, then Exercise 11 allows us to factor $n$.

The question is, how do we find a multiple of $p-1$? One approach may be to hope that $p-1$ is only divisible by the $l$ smallest primes $\ell_1, \ell_2, \ldots, \ell_l$. Since the largest prime power that could divide the (unknown) value $p-1$ is $\ell_i^{\lfloor \log n / \log \ell_i \rfloor}$, we could construct a multiple using

$$k = \prod_{i=1}^{l} \ell_i^{\lfloor \log n / \log \ell_i \rfloor}.$$

In practice, $k = \ell_l!$ will work just as well.

*Exercise* 30. Using the above approach, factor 1829.

We arrive at the following requirement.

**Requirement 3.** Suppose $n = pq$. Let $k_1$ be the largest divisor of $p-1$, and let $k_2$ be the largest divisor of $q-1$. Then $\min\{k_1, k_2\}$ arithmetic operations should be an infeasible computation.

One simple way to ensure this is to choose $p$ and $q$ as safe primes, that is, such that $(p-1)/2$ and $(q-1)/2$ are also prime.

## 5.3 Pollard's $\rho$

We assume that the reader is familiar with Pollard's $\rho$ method for computing discrete logarithms. Our goal this time is to construct three random-looking sequences of integers, one of which will collide and provide us with our factorization.

Let $s_1$ be an integer between 0 and $n$. We construct a sequence of integers $s_1, s_2, \ldots$ using the rule

$$s_{i+1} = s_i^2 + 1 \bmod n. \tag{3}$$

Based on this sequence, we define a second sequence $s_{q,1}, s_{q,2}, \ldots$, where $s_{q,i} = s_i \bmod q$. Note that

$$s_{q,i+1} = s_{q,i}^2 + 1 \bmod q. \tag{4}$$

**Lemma 9.** *Let the sequences $s_1, s_2, \ldots$ and $s_{q,1}, s_{q,2}, \ldots$ be as above. Suppose indexes $i, j$ exist such that $s_{q,i} = s_{q,j}$. Then $\gcd(s_i - s_j, n) > 1$. If $s_i \neq s_j$, then*

$$\gcd(s_i - s_j, n) = q.$$

*Proof.* If $s_i = s_j$, then $\gcd(s_i - s_j, n) = n$, so suppose $s_i \neq s_j$.

Since

$$s_i \equiv s_{q,i} \equiv s_{q,j} \equiv s_j \pmod q,$$

we see that $q$ divides the difference $s_i - s_j$. But $s_i \neq s_j$, so we cannot also have that $p$ divides $s_i - s_j$, and the claim follows. □

*Exercise* 31. In this exercise, we will use our knowledge of the factors of $n$ to better see what is happening. Let $n = 2573 = 31 \cdot 83$ and $s_1 = 2380$.

a. Compute the first 15 terms of the sequences from (3) and (4).

b. Find by inspection the first repetition in the sequence $s_{q,1}, s_{q,2}, \ldots$.

c. Use Lemma 9 and the repetition found to factor $n = 2573$.

**Proposition 10.** *Let $s_1, s_2, \ldots$ and $s_{q,1}, s_{q,2}, \ldots$ be defined as above. Suppose $k$ is the smallest integer such that $s_{q,k} = s_{q,k'}$ for some $k' < k$. Then distinct indexes $i, j$ can be found such that $\gcd(s_i - s_j, n) > 1$ using at most $9k$ arithmetic operations and $k$ gcd computations.*

*Proof.* We consider the sequences $t_1, t_2, \ldots$ and $t_{q,1}, t_{q,2}, \ldots$ given by $t_j = s_{2j}$ and $t_{q,j} = s_{q,2j}$. It is clear that for some $i$, $t_{q,i} = s_{q,i}$, and that this $i$ is at most $k$.

We can compute successively the pairs $(s_1, t_1), (s_2, t_2), \ldots$ using the rule

$$(s_{i+1}, t_{i+1}) = (s_i^2 + 1 \bmod n, (t_i^2 + 1)^2 + 1 \bmod n).$$

By computing $\gcd(s_i - t_i, n)$ we will notice when this is larger than 1, which it by Lemma 9 will be for some $i \leq k$. Computing each new pair requires 9 arithmetic operations. □

*Exercise* 32. The proof of Proposition 10 essentially describes an algorithm that eventually computes a greatest common divisor larger than 1. Write out this algorithm carefully and restate Proposition 10 as a statement about the algorithm's time complexity (in terms of arithmetic operations and gcd computations).

Also, suppose the elements $s_1, s_2, \ldots, s_k$ are all distinct. Show that then the greatest common divisor eventually computed is a proper divisor of $n$.

*Exercise* 33. Use the algorithm from Exercise 32 to factor (by hand) $n = 2573$.

Let $E$ be the event that the $L$ first elements of $s_{q,1}, s_{q,2}, \ldots$ are all distinct, and let $E'$ be the event that the $L$ first elements of $s_1, s_2, \ldots$ are all distinct. Then we can define two functions

$$\theta(L, n) = \Pr[E] \qquad \text{and} \qquad \gamma(L, n) = 1 - \Pr[E'].$$

We can now prove the following result.

**Theorem 11.** *Let $n$ be a product of two distinct primes $p$ and $q$. Then a proper factor of $n$ can be computed using at most $9L$ arithmetic operations and $L$ gcd computations, except with probability $\theta(L, n) + \gamma(L, n)$.*

*Proof.* Some integer will appear twice among the $L$ first elements of the sequence $s_{q,1}, s_{q,2}, \ldots$ except with probability $\theta(L, n)$.

There will be no repetitions among the $L$ first elements of the sequence $s_1, s_2, \ldots$ except with probability $\gamma(L, n)$.

By Exercise 32 there is then an algorithm that computes a proper factor of $n$ using at most $9L$ arithmetic operations and $L$ gcd computations. The claim follows. □

Now suppose the two sequences are "random-looking" with respect to repetitions. This means that since the elements of the second sequence come from a much smaller set, we expect a repetition in that sequence long before we have a repetition in the larger sequence. It seems plausible that the sequences we have defined above are "random-looking" with respect to repetitions. Therefore, we make the following conjecture.

**Informal conjecture 12.** *The function $\theta(L, n)$ is roughly similar to*

$$\exp\left(-\frac{L(L-1)}{2q}\right)$$

*and $\gamma(L, n)$ is roughly similar to*

$$\frac{L(L-1)}{2n}.$$

Based on Conjecture 12 and Theorem 11, we arrive at the following requirement.

**Requirement 4.** Suppose $n = pq$, with $q < p$. Then $\sqrt{q}$ arithmetic operations should be an infeasible computation.

## 5.4 Index Calculus

Index calculus for factoring is very similar to index calculus for discrete logarithms. We begin with the observation that knowledge of more than two square roots modulo $n$ of the same number allows us to factor $n$.

**Proposition 13.** *Let $n$ be the product of two distinct, large primes $p$ and $q$. Let $z$ be a square modulo $n$ with $\gcd(z, n) = 1$. Then $z$ has four square roots modulo $n$.*

*Suppose further that $x$ and $y$ are square roots of $z$ modulo $n$ satisfying*

$$x \not\equiv \pm y \pmod{n}.$$

*Then $\gcd(x - y, n)$ is a proper divisor of $n$.*

*Proof.* If $z$ is a square modulo $n$, then $x$ exists such that $x^2 \equiv z \pmod{n}$, which means that $x^2$ is congruent to $z$ modulo both $p$ and $q$. It follows that $x$ and $-x$ are square roots of $z$ modulo both $p$ and $q$, and they are distinct since $z$ is relatively prime to $n$. The Chinese remainder theorem then says that these can be combined into four square roots of $z$ modulo $n$.

Since $x^2 \equiv z \equiv y^2 \pmod{n}$, we know that $n$ divides $x^2 - y^2 = (x - y)(x + y)$. But since $x \not\equiv \pm y \pmod{n}$, we know that $n$ does not divide $x - y$ and $x + y$. It follows that $\gcd(x - y, n)$ is a proper divisor of $n$. □

The next idea is that if we have a sufficient number of relations between random integers and small primes modulo $n$, then linear algebra will allow us to construct a square root of a product of these random integers.

**Proposition 14.** *Let $t_1, t_2, \ldots, t_{l+1}, \ell_1, \ell_2, \ldots, \ell_l$ be integers satisfying*

$$t_i \equiv \prod_{j=1}^{l} \ell^{s_{ij}}. \tag{5}$$

*Then using at most $(l + 1)^3$ arithmetic operations, we can compute $\alpha_1, \alpha_2, \ldots, \alpha_{l+1} \in \{0, 1\}$ (not all zero) such that*

$$\prod_{i=1}^{l+1} t_i^{\alpha_i} \equiv \left( \prod_{j=1}^{l} \ell_j^{\frac{1}{2} \sum_{i=1}^{l+1} \alpha_i s_{ij}} \right)^2 \pmod{n}.$$

*Proof.* Let $S$ be the $(l+1) \times l$ matrix $(s_{ij})$, where each of the relations defines one row. If we consider $S$ as a matrix modulo 2, it has rank at most $l$, so there exists a non-zero vector $\vec{\alpha}$ such that $\vec{\alpha} S \equiv \vec{0} \pmod{2}$.

Given such a vector $\vec{\alpha}$, we know that the sums $\sum_{i=1}^{l+1} \alpha_i s_{ij}$ are divisible by 2, which means that raising each $\ell$ to these sums divided by 2 gives us a square root of $\prod_{i=1}^{l+1} t^{\alpha_i}$.

Gaussian elimination will find a vector in the kernel of $S$ using at most $(l + 1)^3$ arithmetic operations. □

Finally, if we already know squares of the random integers from our relations, we can easily find a square of the product. This second square will be independent of the one we found above. Which means that we will be able to factor $n$ half the time.

**Theorem 15.** *Suppose $n$ is a product of two distinct large primes, and that a fraction $\sigma$ of the squares modulo $n$ are divisible only by the small primes $\ell_1, \ell_2, \ldots, \ell_l$.*

*Then we can find a proper factor of $n$ with probability $1/2$ using an expected*

$$\sigma^{-1}(l+1)(l+\log_2 n+2)+(l+1)^3+2l^2+2(l+1)\log_2 n+2l$$

*arithmetic operations and one gcd computation.*

*Proof.* Our goal is to construct two independent square roots modulo $n$ of the same number.

We begin by choosing random numbers $r$ between 0 and $n$ for each number checking if $t = r^2 \bmod n$ factors as a product of the small primes. In this way, we eventually find $l+1$ relations of the form (5).

By Proposition 14, we can find $\alpha_1, \alpha_2, \ldots, \alpha_{l+1} \in \{0,1\}$ such that

$$\left(\prod_{i=1}^{l+1} r_i\right)^2 \equiv \left(\prod_{j=1}^{l} \ell_j^{\frac{1}{2}\sum_{i=1}^{l+1}\alpha_i s_{ij}}\right)^2 \pmod{n}.$$

In other words, we have two square roots modulo $n$ of the same number. Since the coefficients $s_{ij}$ depend only on the square modulo $n$ of the random numbers $r_1, \ldots, r_{l+1}$, the two square roots are also independent.

By Proposition 13 we can then factor $n$ with probability $1/2$ using one gcd computation.

For each random number $r$, squaring modulo $n$ requires 2 arithmetic operations. Checking if the square factors as a product of $\ell_1, \ldots, \ell_l$ requires at most $\lfloor l+\log_2 n \rfloor$ arithmetic operations.

We expect to find $l+1$ relations after trying $\sigma^{-1}(l+1)$ random numbers, which means that we need at most

$$\sigma^{-1}(l+1)(l+\log_2 n+2)$$

arithmetic operations to generate the relations. We then need at most $(l+1)^3$ arithmetic operations to find $\alpha_1, \ldots, \alpha_{l+1}$. (Strictly speaking, we work with a binary matrix, so these arithmetic operations are much faster.)

Finally, we need at most $2l^2$ arithmetic operations to compute the sums $\sum_{i=1}^{l+1}\alpha_i s_{ij}$, then at most $2(l+1)\log_2 n$ arithmetic operations to compute the small primes raise to these sums. Finally we require at most $2l$ multiplications to compute the two independent square roots. □

Note that if we do all these calculations and fail to factor $n$, then we do not have to all the work over again. In practice, discarding a few of our old relations and replacing

them by a few new ones will give us another go. In this way, we will quickly factor $n$ without doing significantly more work.

We also expect $l$ to be much larger than $\log_2 n$, so we expect to be able to factor $n$ using approximately

$$\sigma^{-1}l^2 + l^3$$

arithmetic operations. It is reasonable to assume that random squares modulo $n$ are as likely to be products of small primes as random integers. As we have seen, after a suitable choice for $l$, this means that we can factor using approximately

$$\exp\left(\sqrt{8}\sqrt{\log n \log\log n}\right)$$

arithmetic operations.

We have arrived at the following requirement.

**Requirement 5.** $\exp(\sqrt{8}\sqrt{\log n \log\log n})$ arithmetic operations should be an infeasible computation.

Today, we have much better algorithms for factoring. While we shall not study these algorithms, we note that the above requirement is not the final requirement.

# 6 The Public Key Infrastructure Problem

As we have seen, we can construct plausible cryptosystems where anyone who knows the encryption key can encrypt messages, but only those who know the decryption key can decrypt messages.

One problem remains. Alice wants to send a message to Bob. How does she get Bob's encryption key? Suppose Alice finds a key that she thinks belong's to Bob, but which in reality belongs to Eve who has the corresponding decryption key. Eve can then easily decrypt Alice's ciphertext.

One possible solution is a public key directory, modeled on a telephone directory, listing people and their public keys. It seems impractical to have printed copies of this directory, so it would have to be an online service, which begs the question: How can Alice be sure that the encryption key she just fetched came from the directory, and not from Eve?

The *public key infrastructure* problem is Alice's problem of getting hold of Bob's public key.

Another interesting problem is what happens when Bob receives Alice's ciphertext. How does he know that it comes from Alice, and not from Eve?

It turns out that both of these problems have reasonable solutions involving digital signatures.