Lecture Notes

edX Quantum Cryptography: Week 5

Distributing Keys

Contents

This week, we're finally distributing actual keys! We will approach our objective in a series of steps, ending up with the famous BB84 quantum key distribution (QKD) protocol. Throughout, we will assume that Alice and Bob share an *classical authenticated channel (CAC)*. Whenever Alice and Bob make use of this channel below, we will say they send the information over "the CAC".

Let us now start our investigations into quantum cryptography. Imagine that we have several parties engaging in some communication protocol in order to solve a specific task. For example, the parties may just want to send information, or they may want to search a database, or engage in an auction. The goal of cryptography is to protect the honest parties from the dishonest ones during such an interaction. What does it mean to be honest or dishonest?

Definition 5.0.1 — Honest and dishonest (informal). Given any communication protocol between several parties:

- A party is call *honest* if he/she follows the protocol precisely. That is, the party will give the correct input to the protocol, execute all steps as dictated, and produce the desired output.
- A party is called *dishonest* or *malicious*, if he/she does not follow the protocol. Instead, this party can deviate from the protocol aribtrarily. A malicious party is also called an *adversary*.

When designing any cryptographic protocol, we first have to ask ourselves what kind of malicious parties we want to protect against. That is, whether there are some limits to what a malicious party can do in order to break the protocol, and how many resources he has at his disposal. A standard assumption that is generally implicit in all cryptographic protocols is that the honest party, let us call her Alice, sits in an impenetrable lab that the adversary does not have control over. In other words, Alice can perform local computations without the adversary's knowledge. Only when Alice sends any information out of her own lab along a communication channel does the adversary have any opportunity to intercept or tamper with the protocol execution. We will see later that quantum information makes it possible to weaken this demand! For the moment, however, we will assume that an honest Alice (or Bob) has full control over her lab.

5.1 Secure key distribution

The main cryptographic challenge that we will consider is the one of key distribution. Here, our protagonists, Alice and Bob want to protect their communication from the prying eyes of an eavesdropper Eve. Alice and Bob are thereby always honest, and Eve is the adversary. Alice and Bob have control over their secure labs that Eve cannot peek into. However, Eve has access to the communication channel connecting Alice and Bob.

Definition 5.1.1 — Key distribution. A *key distribution* protocol between Alice and Bob aims to achieve the following goals, given a security parameter $\varepsilon \geq 0$:

- $\varepsilon-$ correctness: Alice and Bob both agree on an m-bit key $K \in \{0,1\}^m$, except with some failure probability at most ε. That is, both Alice and Bob have K_A, K_B respectively, and $\text{Prob}(K_A \neq K_B) \leq \varepsilon$.
- $\varepsilon-$ security: Any outsider Eve is almost ignorant about the key, i.e. $\rho_{KE}^{\text{real}} \approx_\varepsilon \rho_{KE}^{\text{ideal}}$ where $\rho_{KE}^{\text{ideal}} = \frac{\mathbb{I}_K}{2^m} \otimes \rho_E$.

To achieve such a key distribution protocol, we will consider the following communication channels which Alice and Bob may have access to:

1. A classical channel: Alice and Bob can send classical bits in either direction over this channel. Eve has complete access to this channel. In particular, she can read all messages, modify them, and even impersonate Alice (or Bob).
2. A classical *authenticated* channel (CAC): A classical communication channel with one extra

guarantee: Alice and Bob know that the message originated unaltered from Alice or Bob respectively. This means that while the channel is not secret because Eve can still read all the messages that travel across, she cannot impersonate Alice or Bob or alter messages traveling over the channel.

3. A classical *secret* channel: A classical communication channel in which Eve cannot learn any information (see below!) about the messages traveling across. Yet, while she cannot hope to gain any information about the message, Eve could impersonate Alice or Bob.
4. A classical *secret and authenticated* channel: A classical communication channel combining both guarantees above.
5. A *quantum communication* channel: A channel where Alice may send quantum information (in particular, in the form of qubits) to Bob, where Eve has full access to all the quantum communication.

For simplicity we will start our discussion by assuming that Alice and Bob are already connected by a CAC - we will see later how such a CAC can be built. That is, we will for the moment only worry about establishing a key which is hidden from Eve!

5.2 Distributing keys given a special classical channel

As a warmup, let us consider how we can generate a key from a very special classical channel - which will include many of the essential ideas we will need in quantum key distribution. This special channel has the property that Eve cannot completely intercept all messages going across, but her ability to eavesdrop is somehow *guaranteed* to be limited. For the moment, let us take the special channel to have the following properties: whenever Alice sends a bit $b \in \{0,1\}$ across the special channel (SC) then

- Bob correctly receives b.
- Eve obtains the bit b correctly with probability q, otherwise with probability $1-q$ receives the bit $1-b$ (but she does not know whether the bit is correct or not).

In this special channel, we have modelled noise for Eve's infomation in an explicit manner, where each bit of b^E is obtained from b by flipping it with some probability $1-q$. Such a noise model is called the binary symmetric channel, which we denote as $BSC(q)$.

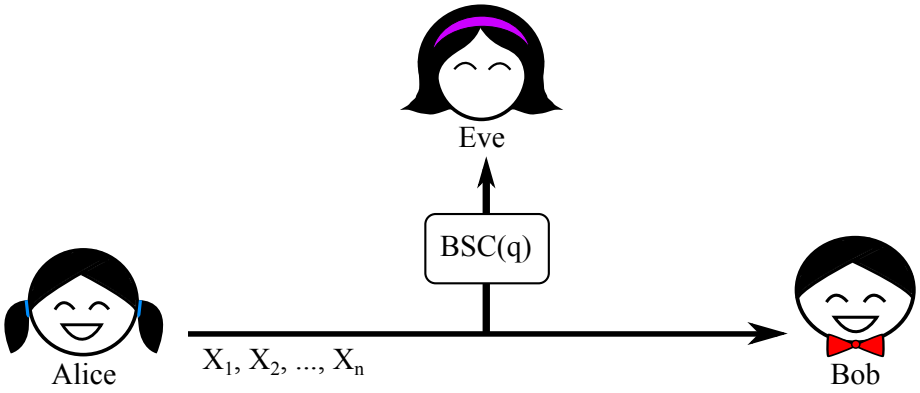

Figure 5.1: Distributing keys over a special classical channel.

Let us now consider the following simple protocol.

Protocol 1 — Key distribution using a special channel. Consider the following protocol:

1. Alice chooses a string $x = x_1, \ldots, x_n \in \{0,1\}^n$ uniformly at random, and sends each bit x_j

to Bob over the special channel.
2. Alice picks a random seed $r \in \{0,1\}^m$, uses a function $Ext : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^l$ and applies the extractor to X, computing $k = Ext(x,r)$.
3. Alice sends r to Bob over the CAC.
4. Bob computes $k = Ext(x,r)$.

Remember that to show that our protocol works we need to establish two things: first, we want that the protocol is ε-correct, that is, Alice and Bob output the same key (except for some small probability of failure). Second, we want to show that the protocol is ε-secure, for some parameter ε. To see that the protocol is correct, note that the special channel is such that Bob receives all bits correctly. That is, he obtains $x = x_1, \ldots, x_n$ without error. Since he also learns r, i.e., he knows which function $Ext(\cdot, r)$ to apply to x, he can compute $k = Ext(x,r)$ without error.

Why would this protocol be secure? Let us first note that Eve's probability of guessing each bit is precisely given by q. We thus have that

$$P_{\text{guess}}(X|E) = q^n \ , \tag{5.1}$$

or equivalently

$$H_{\min}(X|E) = -\log P_{\text{guess}}(X|E) = n(-\log q) \ . \tag{5.2}$$

If r is, for example, used to choose a function $Ext(\cdot, r)$ from a set of two-universal functions, then we are guaranteed that

$$D\left(\rho_{KRE}, \frac{\mathbb{I}}{2^\ell} \otimes \rho_{RE}\right) \leq \varepsilon \tag{5.3}$$

whenever $\ell \leq H_{\min}(X|E) - 2\log(1/\varepsilon) - 1$, as shown in [Ren05]. We have also seen this in the lecture notes last week. Therefore, whenever we generate a key that is at most $\ell \leq n(-\log q) - 2\log(1/\varepsilon) - 1$ bits long with this procedure, then we know that we are ε-secure. In cryptography, we typically fix ε and ℓ in advance, and then ask how large n has to be in order to achieve the desired key length ℓ with the guarantee ε. Note that Eq. (5.3) is to some extend remarkable: even if the eavesdropper *later* learns which function $Ext(\cdot, r)$ we applied, then nevertheless she cannot learn anything about the key! The fact that she learns r only later, however, is of crucial importance. If Eve would know r ahead of time, she could tailor her entire attack to the knowledge of r. In the protocol above this clearly wouldn't matter much at all, since we are given such a strong guarantee on what Eve can do in order to learn the bits, but in general we will have to be content with knowing only the min-entropy without such additional information.

We have seen now that whenever it is guaranteed that Eve cannot intercept all messages perfectly, i.e. there are some noise or losses for her while receiving information from Alice, then the min-entropy $\mathrm{H}_{\min}(X|E)$ will be high (since it is additive over n rounds), and therefore if Bob receives X perfectly, then Alice and Bob can always extract a key which is ε-secure against Eve.

■ **Example 5.2.1** Consider another special channel where Eve receives bits without noise, however Eve has limited memory and hence can only store a maximum number of S bits, that is, $|E| \leq 2^S$. If Alice sends a completely random, n-bit string X across the channel, then $\mathrm{H}_{\min}(X) = n$, and Eve's knowledge about X can be lower bounded by

$$\mathrm{H}_{\min}(X|E) \geq \mathrm{H}_{\min}(X) - \log|E| \geq n - S. \tag{5.4}$$

We thus see that whenever the length of X is greater than Eve's storage, i.e. $n > S$, Alice and Bob can use an extractor to extract a non-zero amount of key which will be secure against Eve. ■

5.3 Information reconciliation

So far, we have always assumed that there are no errors on the channel connecting Alice and Bob. Clearly, this is extremely unrealistic in any real implementation. If we follow the procedure in Protocol 1 when using this channel, the *correctness* of the protocol is affected: Alice and Bob now hold two different strings $X, \tilde{X}$ which are different from each other. Therefore, while performing privacy amplification, they will hash down two different strings and hence (very likely!) end up with two different keys $K_A \neq K_B$ that are useless for further communication.

How can we deal with this problem? The key idea is to perform error-correction. To this end, Alice and Bob need to perform one additional step before privacy amplification called *information reconciliation* in which they exchange error-correcting information to correct errors.

First of all, let's describe the communication scenario and introduce some convenient notation. Alice and Bob hold two strings that we denote X_A and X_B. X_B, the string of Bob, equals X_A plus a string of errors that we denote by S. Alice and Bob are connected by a CAC, and therefore an information reconciliation protocol consists simply in the exchange of messages over this channel in order for Bob to recover X_A. We denote by C the string consisting of all the messages exchanged over the classical channel. Finally, Bob, with the help of X_B and C will output $\hat{X}_A$, that is, an estimate of the string of Alice X_A.

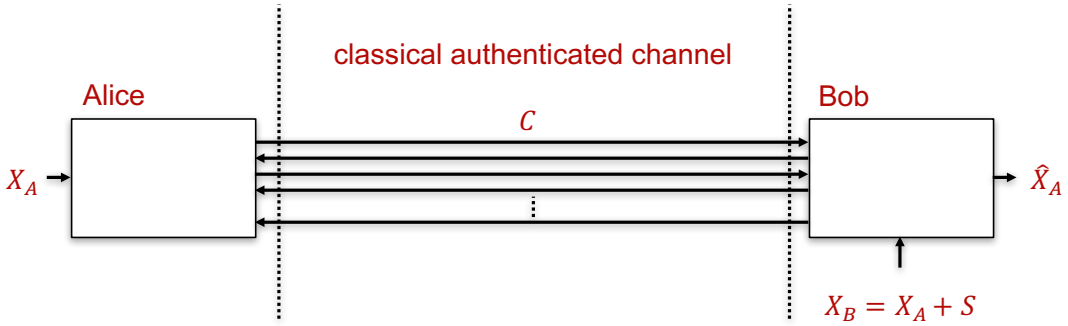

Figure 5.2: Scheme of a generic information reconciliation protocol.

Definition 5.3.1 — Information Reconciliation. Let X_A, X_B be distributed according to the joint probability distribution $P_{X_A X_B}$. An *information reconciliation* protocol for X_A, X_B is ε-correct and leaks $|C|$ bits if :

- $\text{Prob}(X_A \neq X_B) \leq \varepsilon$.
- The length of the messages exchanged on the public channel is $|C|$.

The goals of the information reconciliation step are two-fold. First and most obvious, Alice and Bob want to ensure that after reconciliation the strings X_A and $\hat{X}_A$ are ε-correct, that is, that the probability that they are different is at most ε. Second, note that all classical communication between Alice and Bob is public, which means that Eve can also gain information from the error correction information they send across! Again recalling the chain rule for the min-entropy, we have

$$H_{\min}(X|EC) \geq H_{\min}(X|E) - |C| \,, \tag{5.5}$$

where we used C to denote the error-correcting information sent across the classically authenticated channel. We thus have the min-entropy of Eve *with* the error-correction information C can shrink by at most the number of bits $|C|$ of error-correction information that Alice and Bob send.

Again, it is very easy to achieve any of both goals independently. Why is this? Imagine that a reconciliation protocol consists of Alice sending her whole string to Bob over the classical channel.

This is a great protocol if we only care about correctness, the strings will definitely be correct, but the leakage is maximal and after reconciliation we would not have any min-entropy left to do privacy amplification.

On the other hand, imagine a reconciliation protocol that consists in Alice and Bob doing nothing, then, for leakage purposes, the protocol is perfect, the leakage is zero, but the strings might not be correct for the desired ε.

We can classify information reconciliation protocols depending on their usage of the classically authenticated channel. The most general protocol might consist in the exchange of messages in both directions, that is from Alice to Bob and from Bob to Alice. We call such a protocol a two-way or an interactive protocol. However, much more simpler, and in many circumstances, it is already sufficient that the whole reconciliation consists of a single message from Alice to Bob, that is, the communication happens one-way. We refer to such protocols as one-way reconciliation protocols, where Alice encodes her string X_A into a single (significantly shorter) message that we denote by C_A and sending it through the classical channel. Then Bob uses C_A to eliminate errors from X_B, effectively recovering X_A.

5.4 Syndrome coding

In the following we will explore one concrete one-way reconciliation scheme. The scheme that we will describe is based on linear codes. Let us review the definition and main elements of linear codes:

Definition 5.4.1 — Linear code. Let $\mathbb{F}_q$ be the finite field of size q, a (n,k) q-ary linear code C is a linear subspace of $\mathbb{F}_q^n$ of dimension k.

The individual elements (which are q-ary strings of length n) contained in the subspace defining a linear code are called *codewords*, and a (n,k) q-ary code has q^k different codewords. We will only be concerned with binary codes, so in the following we let $q = 2$.

Since a code is just a subspace, if we want to construct an n length binary code, any procedure that characterizes a subspace of $\mathbb{F}_2^n$ will allow us to construct a code. One convenient characterization is by using a $m \times n$-dimensional *parity check matrix* H. The code is defined as the set of vectors v such that $H \cdot v^T = 0$. The dimension of the code induced by H is $k = n - \text{rank}(H)$.

The map $s_H : \mathbb{F}_2^n \to \mathbb{F}_2^m$ given by $v \mapsto H \cdot v^T$ is called the *syndrome*. It turns out that the syndrome is very useful beyond inducing the code. In particular, an example of a reconciliation procedure is when Alice and Bob agree on a particular parity matrix H. Let's consider this in more detail. First, let us discuss the encoding step performed by Alice. The reconciliation scheme is based on linear codes, given a parity check matrix H and Alice's vector X_A, the encoding of X_A is its syndrome. That is, the message that Alice sends to Bob for information reconciliation is the syndrome of X_A.

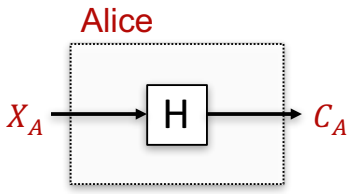

Figure 5.3: The encoder in syndrome coding based reconciliation.

■ **Example 5.4.1** Let $v = (001)$ and $H = \begin{pmatrix} 110 \\ 011 \end{pmatrix}$. Then $s_H(v) = H \cdot v^T = (01)^T$. What this means,

is that if v were the string of Alice, the message that Alice would send to Bob for reconciliation would be $s_H(v)$ which as we calculated is: $(01)^T$. ∎

Let us now move to Bob, who has a decoder, which is essentially a procedure that helps him to estimate the error string S (where recall $X_B = X_A + S$), so that he may recover X_A. The decoder is a little bit more complicated than the encoder, since it takes both C_A and X_B as inputs. The first thing that happens in the decoder is that it computes the syndrome of X_B that we call C_B. Then, Bob computes $C_S = C_B + C_A$, where recall that C_A is the syndrome of X_A. We call this resulting string C_S because it is indeed the syndrome of the error string, i.e. $C_S = s_H(S)$. Then, C_S is sent into a module that estimates the error string S and outputs the estimate that we call $\hat{S}$. Finally $\hat{S}$ is added to X_B and this will be the decoded string that Bob will receive.

Exercise 5.4.1 Show that C_S is indeed the syndrome of the error string S, or in other words, that $s_H(S) = s_H(X_A) + s_H(X_B)$. ∎

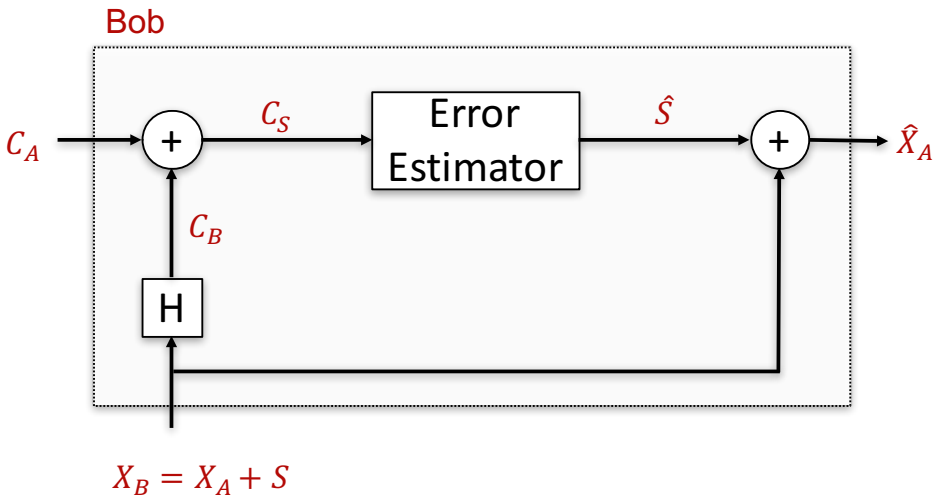

Figure 5.4: The decoder in syndrome coding based reconciliation.

Exercise 5.4.2 Show that if the error estimate is correct $\hat{X}_A$ will equal X_A. ∎

A simple but relevant property of the scheme is that as soon as the length of the string that Alice sends Bob is smaller than the length of the string of Alice not all errors can be corrected. In order to see this recall that $\hat{X}_A$ is $X_B + \hat{S}$, but $\hat{S}$ is a function of C_S, the syndrome of S. Hence, even if the estimator function outputs a different value for each syndrome we have 2^m different outputs while there are 2^n different error strings, in other words, unless m equals n, it is not possible to correct all errors. However, if different errors occur with different probabilities we might be satisfied if we correct the most likely errors.

∎ **Example 5.4.2** Let us go back to Example 5.4.1. Let us now describe the decoder, for this we need to make explicit the estimation function. There are four different syndromes. We will assign as error estimate for each syndrome the following strings:

Syndrome	Error Estimate
00	000
01	001
10	100
11	010

Can you guess why we chose this particular map? The idea is that if there is zero or one errors, the estimator will output the correct error estimate. However, this also means that any other error string will be wrongly estimated. ∎

5.5 Limits of reconciliation

We have seen a concrete scheme for reconciliation. What are the fundamental limits of reconciliation? In order to answer this question we need some structure. Let us assume that the strings of Alice and Bob, that we denote for precision X_A^n and X_B^n, are of length n, where each of the symbols is drawn independently from the same joint distribution $P_{X_A X_B}$ (where X_A, X_B are binary random variables). Then, any information reconciliation protocol that leaks $|C|$ bits satisfies:

$$|C| \geq n \cdot H(X_A|X_B) \tag{5.6}$$

Moreover, the inequality can be achieved when $n \to \infty$ [SW73].

In a realistic scenario n is finite and the information reconciliation protocol needs to be computationally efficient. Instead of dealing with the implementation details of an information reconciliation protocol, it is sometimes convenient to approximate the leakage value of a realistic protocol by $\xi \cdot nH(X_A|X_B)$, where $\xi > 1$ is the reconciliation efficiency. The constant ξ is often chosen $\xi \approx 1.2$. However, this approximation should be used with care since ξ will be a function of the length, the noise model and the correctness considered [Tom+14].

The errors between Alice's and Bob's string can also generally be modelled by a BSC. That is, we can see their strings as the input and output of a BSC: whenever Alice inputs a bit x, Bob receives x with probability p, but with probability $1-p$ the bit is flipped to $x+1$[1]. In the case of a BSC, Eq. (5.6) simplifies to $|C| \geq nh(p)$ where $h(p) = -p\log(p) - (1-p)\log(1-p)$ is the binary entropy function.

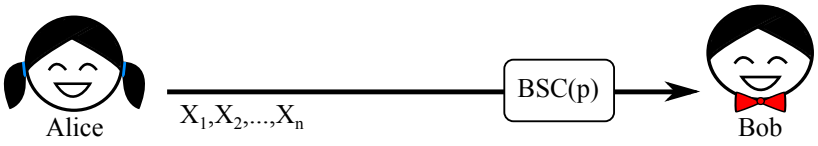

Figure 5.5: The errors between Alice's and Bob's string are generally modelled by a BSC.

5.6 Further reading

So let us conclude this module on reconciliation with some considerations.

The example that we described in Examples 5.4.1 and 5.4.2, works but it does not scale up, if we try to extend the same ideas to a parity check matrix of m by n, we will need a table with 2^m entries to decide the output of the estimator module. However, the method of reconciliation based

[1] In the context of quantum key distribution (QKD), this error probability $1-p$ is also often called the quantum bit-error rate (QBER).

on syndrome can be adapted to use any family of linear codes. Previous work has obtained leakage close to the theoretical optimal using LDPC codes, polar codes, turbo codes, etc.

Previous to this idea of using linear error correcting codes and one-way reconciliation, there was some ad-hoc two-way protocol, proposed specially for information reconciliation. Its name is Cascade [BS93], it has a reasonably simple description and it is not very difficult to implement. However, the original version was suboptimal from the point of view of leakage. Fortunately, there are some nice modifications that make it competitive with the error correcting codes based reconciliation.

Acknowledgments

 This chapter of the lecture notes was written by David Elkouss, Nelly Ng, Thomas Vidick and Stephanie Wehner. We thank Kenneth Goodenough, Jonas Helsen, Jérémy Ribeiro, and Jalex Stark for proofreading.

Bibliography

[BS93] Gilles Brassard and Louis Salvail. “Secret-key reconciliation by public discussion”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1993, pages 410–423 (cited on page 10).

[Ren05] R. Renner. “Security of Quantum Key Distribution”. PhD thesis. ETH Zurich, 2005 (cited on page 5).

[SW73] David Slepian and Jack Wolf. “Noiseless coding of correlated information sources”. In: *IEEE Transactions on information Theory* 19.4 (1973), pages 471–480 (cited on page 9).

[Tom+14] Marco Tomamichel et al. “Fundamental finite key limits for information reconciliation in quantum key distribution”. In: *2014 IEEE International Symposium on Information Theory*. IEEE. 2014, pages 1469–1473 (cited on page 9).