6 Distributing keys

This week, we're finally distributing keys! We will approach this objective in a series of steps, ending up with the famous BB'84 quantum key distribution (QKD) protocol in the next chapter and its security analysis in the next chapter and the following one. Before we describe the task of key distribution in detail, let's first discuss the meaning of "honest" and "dishonest" parties in a cryptographic protocol.

6.1 Honest and Dishonest

Imagine that several parties engage in some communication protocol. For example, the parties may want to send each other some information, or one of them may want to search a database that is in the possession of another, or they may want to engage in an auction for some public goods, etc. 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 6.1.1 (Honest and dishonest (informal)) *A* communication protocol *between several parties specifies the actions that each party is intended to take in the protocol. These actions may depend on an* input *that the party is supposed to have, and each party may produce an* output *at the end of the protocol.*

- *A party is called* honest *if he/she follows the protocol precisely. That is, the party initializes their part of the protocol in the correct way, executes all steps as dictated, and returns an output as expected.*
- *A party is called* dishonest *or* malicious*, if he/she does not follow the protocol. Instead, they can deviate from the protocol in an arbitrary way. This includes for example refusing to answer a question, sending some unsolicited information, etc. A malicious party is sometimes called an* adversary.

When designing a cryptographic protocol we have to ask ourselves what kind of malicious parties the protocol should protect against. In other words, whether there are some limits to what a malicious party can do in order to break the protocol, and how many resources he or she has at their disposal. To be precise we should say "what are the limits," and not "whether there are limits," because there are always limits—for example, if the adversary is allowed to corrupt all users in the protocol then surely no meaningful security can be obtained. Concretely, a minimal assumption that is often left implicit in cryptographic protocols is that a 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 information out of her own lab along a communication channel with another party does the adversary have an opportunity to intercept or tamper with the protocol execution. Here we make that assumption as well; however, in Chapter **??** we will see that by making use of quantum information makes it is possible to weaken that demand.

6.2 Secure key distribution

The main cryptographic task that we consider in this book is the one of key distribution. In this task there are two honest protagonists, Alice and Bob, and one malicious eavesdropper, Eve. Alice and Bob each have secure labs that they have control over and Eve cannot peek into. Moreover, they have access to a variety of communication channels. However, in general Eve may also have access to these channels. A key distribution protocol is a collection of actions for the honest users Alice and Bob such that at the end of the protocol two conditions should hold. First of all, if all goes well Alice and Bob each possess the same key K, which is simply some string of bits. Second, no one else than them should have any information about K. This "no one else" is modeled by the eavesdropper Eve (which can behave as she likes), and we require that from the point of view of Eve the key K appears uniformly random.

Let's make these requirements more precise in order to arrive at a formal definition of security for quantum key distribution that we can work with. We consider the two requirements in turn. The first requirement is called *correctness*: whatever Alice and Bob return, call it K_A for Alice and K_B for Bob, these should always be the same: $K_A = K_B$. As it turns out this is too strong of a requirement, for two reasons. First of all we should always allow a chance that the protocol *aborts*. For example, any Eavesdropper can completely block the communication channels. In this case Alice and Bob would be forced to "give up". So we can only require that *whenever Alice and Bob both return a key K_A and K_B, then these keys are the same*. To model the case where they abort we introduce a new symbol $\perp$; so if $K_A = \perp$ or $K_B = \perp$ then this means that Alice or Bob decided to abort, respectively. (They do not necessarily both always abort at the same time, as Alice could suspect something suspicious but fail in communicating this to Bob, for example because Eve is interfering with the communication channel.) Even this condition is too strong unfortunately. This is because even for a very good protocol we need to allow a chance that things go wrong and yet the honest users don't detect it. We will use a parameter ε to denote the probability that this happens, and strives to design protocols that get the smallest possible ε. We can now give the first half of the formal security definition.

Definition 6.2.1 (ε_c-correctness) *A key distribution protocol between Alice and Bob is* ε_c-correct *if the following holds. Letting K_A and K_B denote the user's outcomes in the protocol, then*

$$\text{Prob}(K_A \neq \perp \wedge K_B \neq \perp \wedge K_A \neq K_B) \leq \varepsilon_c \, .$$

Next we consider the *secrecy* requirement. This is the requirement that "Eve has no information about the key". How do we formalize such an assumption? Observe that at the end of the protocol we can always write the joint state of the key K_A and Eve's quantum state as a CQ state $\rho_{K_A E}$. If Eve has no information about K_A, as we saw in Chapter **??** this means that $\rho_{KA} = 2^{-\ell}\mathbb{I}_K \otimes \rho_E$, where ℓ is the length of K in bits. We call this state the "ideal" state. Now, as for correctness it would be unrealistic to require that the final state of the protocol is always exactly the ideal state. First of all, we can only require that it is very close: ε-close in trace distance. Secondly, we can only require that this is the case when the Eavesdropper is not doing something too crazy, i.e. when the chance that she makes the protocol abort is not too high.[1] Here is the formal definition.

Definition 6.2.2 (ε_s-secrecy) *A key distribution protocol is* ε_s-secret *if the following hold. Let* $\Pr(\textit{abort})$ *denote the probability that either Alice or Bob returns $\perp$. Then it should be the case that*

$$(1 - \Pr(\textit{abort}))\left\|\rho_{KE}^{\text{real}} - \rho_{KE}^{\text{ideal}}\right\|_1 \leq \varepsilon_s \, , \tag{6.1}$$

[1] This is because we can always consider an eavesdropper that for example would always force the protocol to abort unless some very specific conditions are satisfied, which would somehow guarantee that the key is some fixed value such as 0^ℓ.

where ρ_{KE}^{real} is the joint state of Alice's output K_A and the Eavesdropper in an execution of the protocol and $\rho_{KE}^{\text{ideal}} = \frac{\mathbb{I}_K}{2^m} \otimes \rho_E$.

Observe that (6.1) is equivalent to saying that, either $\Pr(\text{abort})$ is very close to 1, in which case the equation is satisfied, or it is not very close to 1 and in that case we require that $\rho_{KE}^{\text{real}} \approx \rho_{KE}^{\text{ideal}}$. Although we do not discuss composable security here (see Section 6.6) we note that it is possible to arrive at the security definitions above by taking a more general perspective of "composable security". In particular the definitions guarantee that the key output by Alice and Bob remains secure if the key distribution protocol is used as an ingredient in a much larger cryptographic protocol.

Definition 6.2.2 refers to "the Eavesdropper." As mentioned in the previous section it is quite important to clarify exactly what is the power that this Eavesdropper may have. We make the following assumptions:

1 All parties are bound by the laws of quantum physics. Even though this may seem obvious, it is worth stating explicitly. In the end of the day our security proof will model all possible actions of the Eavesdropper, and the framework we will use for this is quantum mechanics. If quantum mechanics is wrong or incomplete, our security proof may not hold against adversaries that make use of unexpected physical effects. In particular, we don't consider relativistic effects, black holes, and the like. (Luckily– we'd need quite a few more books to set these up!)
2 The users Alice and Bob behave honestly, i.e. as described in the protocol. They have access to private labs that are perfectly shielded from the Eavesdropper. All computations performed in their labs, classical or quantum, is done perfectly. This includes the generation of random numbers, preparation of qubits, measurements, etc., whenever required by the protocol. (Later we will discuss a weakening of this requirement where the qubit preparation and measurement devices may make small errors.)
3 The Eavesdropper has access to all communication that takes place between the user, and can intercept and modify messages at will (with one important limitation: see the description of the authenticated channel below). In addition the Eavesdropper may make use of an arbitrarily large classical or quantum computer.

We now make more precise the second assumption by describing the type of communication channel which Alice and Bob may have access to in the protocol. Here are some channel types that we may consider. For each type of channel, we describe what access the Eavesdropper has to communication made over that channel.

1 A classical channel: Alice and Bob can send classical bits in either direction over this channel. Eve has complete access to the channel. In particular, she can read all messages, copy them, modify them, and even impersonate Alice (or Bob).
2 A classical *authenticated* channel (CAC): A classical communication channel with one extra guarantee: For any message sent on that channel, Alice and Bob are promised that the message originated from Bob or Alice respectively and moreover that it has not been altered in any way. This channel is not secret, because Eve can still read all the messages that travel on it, but Eve 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 about the messages traveling over the channel. Yet, while she cannot hope to gain any information about any messages sent by the users, Eve could impersonate them to send fake messages (or delete or replace messages that they send, without reading them).
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 and vice-versa, such that Eve has full access to all the quantum communication.

Concretely, the protocols that we consider in this chapter and the following ones will always assume that Alice and Bob have access to (1) a classical authenticated channel and (2) a quantum communication channel. The assumption that the classical channel is authenticated is an important one and we discuss it in more detail in Section 6.5. Before we get to the real protocols, let us start by making our life easier by assuming that Alice and Bob have access to some special kind of communication channels such that the Eavesdropper's access is further restricted.

6.3 Distributing keys given a special classical channel

In this section we consider how Alice and Bob can generate a key from a very special classical channel, as this includes many of the essential ideas we will need to analyze our real quantum key distribution protocols in the next chapters. The special channel that we consider here has the property that Eve cannot completely intercept all messages going across: her ability to eavesdrop is somehow *guaranteed* to be limited. Let $0 \leq q \leq 1$ be a parameter. The channel is called *binary symmetric channel*, which we denote as $BSC(q)$. It has the following properties: whenever Alice sends a bit $b \in \{0,1\}$ across the $BSC(q)$,

- Bob correctly receives b and
- 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).

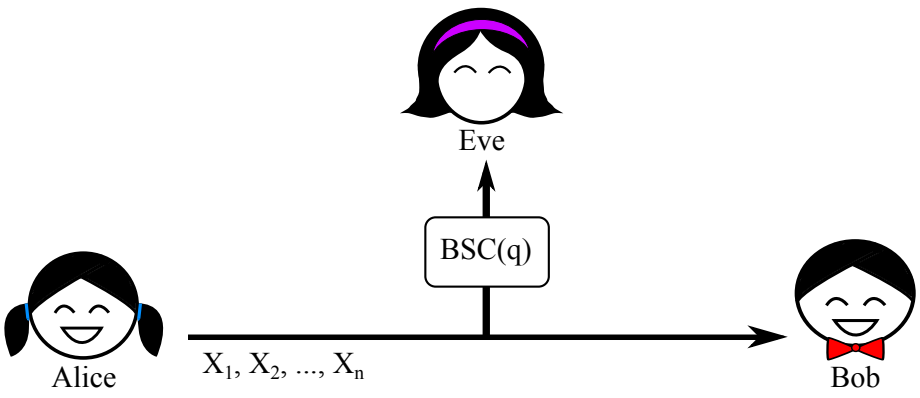

Fig. 6.1 Distributing keys over a special classical channel.

Can we design a correct and secure key distribution protocol using such a channel? At least for some values of q the answer is yes. For example, if $q = \frac{1}{2}$ then you can see that Even always learns a uniformly random bit, i.e. nothing, and so a secure key can be exchanged directly. What about other values of q? Let's estimate how much information Eve can gain about a message sent by Alice to Bob on the $BSC(q)$. So suppose that Alice chooses a message $x = x_1, \ldots, x_n \in \{0,1\}^n$ uniformly at random and sends it to Bob. By definition of $BSC(q)$, Eve will obtain a string $e \in \{0,1\}^n$ such that on average about qn entries are correct and $(1-q)n$ entries have been flipped. Bob, however, receives x exactly. So Alice and

Bob have the same string, but Eve has some amount of information about it which can be quantified as a function of the parameter q. In this situation, how can Alice and Bob *extract* a secure key out of their partially secret common information?

If you didn't read Chapter **??** on your way here, now might be a good time to do so! In that chapter we introduced the task of privacy amplification, which is precisely what Alice and Bob have to do here. Moreover, we also gave a method to solve privacy amplification: apply a randomness extractor! This suggests the following simple protocol.

Protocol 1 (Key distribution using a binary symmetric channel) *Consider the following protocol. Let q be the parameter of the BSC. Let integers n, t, ℓ be chosen such that* (6.5) *is satisfied. Let $Ext : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ be a two-universal randomness extractor (see Section* **??***). In the 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, $j = 1, \ldots, n$, to Bob over the $BSC(q)$.*
2 *Alice picks a uniformly random seed $r \in \{0,1\}^t$ and computes $k_A = Ext(x, r)$.*
3 *Alice sends r to Bob over the CAC.*
4 *Bob computes $k_B = Ext(x, r)$.*
5 *Alice returns k_A and Bob returns k_B.*

Remember that we need to establish two things for this to be a valid quantum key distribution protocol. 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. 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, his value $k_B = Ext(x, r)$ is such that $k_A = k_B$ with certainty. So, this protocol is 0-correct.

Why would the protocol be ε-secure? Let us first note that by definition Eve's probability of guessing each bit correctly is precisely given by q. Let's also assume that $q > \frac{1}{2}$, so Eve gets the correct value with probability more than $1/2$. (We can always reduce to this case by deterministically flipping each bit received by Eve.) In this case Eve's best guess for the real bit is clearly the value that she obtained. Thus if X_i is the random variable associated with Alice's i-th bit, and E_i the value received by Eve, then $P_{\text{guess}}(X_i|E_i) = q$. Since all the bits are chosen and communicated independently,

$$P_{\text{guess}}(X|E) = q^n \,, \tag{6.2}$$

which can be rewritten as

$$H_{\min}(X|E) = -\log P_{\text{guess}}(X|E) = n(-\log q) \,. \tag{6.3}$$

If Ext is, for example, the two-universal extractor from Definition **??** then by Theorem **??** we are guaranteed that

$$D\left(\rho_{KRE}, \frac{\mathbb{I}}{2^\ell} \otimes \rho_{RE}\right) \leq \epsilon \tag{6.4}$$

whenever $\ell \leq H_{\min}(X|E) - 2\log(1/\varepsilon) - 1$. Therefore, whenever we choose the parameter ℓ such that

$$\ell \leq n(-\log q) - 2\log(1/\varepsilon) - 1 \tag{6.5}$$

we obtain a protocol that is 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; here, we get that $n = \ell/\log(1/q) + \Omega(\log(1/eps))$ suffices.

We pause to notice that for the second phase of the protocol, when Alice sends the seed r to Bob, we

did not make use of the $BSC(q)$. Instead, we used the CAC, which means that Eve can entirely learn the seed r. Yet we still obtain security even under that assumption, thanks to the guarantee provided by Eq. (6.4), which includes the key K: this equation means that even if the eavesdropper holding system E learns which function $Ext(\cdot, r)$ is applied, then nevertheless they cannot learn anything about the key (up to ε)! The fact that she learns r only later, after having obtained the system E, is crucial. If Eve would know r ahead of time, then in general she could tailor her entire attack to the knowledge of r. In our scenario where we fix exactly what Eve gets then this doesn't really matter, but when we see the real protocols it will be important that r is only determined after the information from which the key will be created has been exchanged.

Exercise 6.3.1 Consider what happens if Eve gets the bit with probability q, but knows whether her intercept attack was successful. (With the remaining probability $1 - q$, she gets a special symbol $\star$ indicating that the bit was lost.) If we fix ε and n, can you obtain a longer or shorter key in this case?

The way we used the extractor in the protocol is general: we see that whenever the protocol is such that it is possible to show that the min-entropy $\mathrm{H}_{\min}(X|E)$ must be high, and moreover such that Bob has the same information as Alice, then Alice and Bob can always extract a key which has length $\approx \mathrm{H}_{\min}(X|E) - O(\log(1/\varepsilon))$ which is ε-secure against Eve.

Example 6.3.1 *Consider another special channel where all the information that Alice sends automatically goes to Eve,* except *that Eve has limited memory and can only store a maximum of S bits in total. 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 is*

$$\mathrm{H}_{\min}(X|E) \geq \mathrm{H}_{\min}(X) - \log|E| \geq n - S\,, \tag{6.6}$$

where the first inequality is by Exercise **??**. *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 secure key.* ■

6.4 Information reconciliation

In the examples from the previous section we assumed that there are never any errors on the channel connecting Alice and Bob. Clearly, this is extremely unrealistic in any real implementation. If we follow Protocol 1 when using a noisy channel the *correctness* of the protocol will be affected: at the end of the first step Alice and Bob will hold two strings $x_A \neq x_B$ which are not always the same. When applying the extractor to perform privacy amplification, there is a priori no reason for them to magically obtain the same k_A and k_B.

How can we solve this problem? The key idea is to perform a step of *information reconciliation* on the strings x_A and x_B, prior to privacy amplification. In this step Alice and Bob exchange error-correcting information about x_A, x_B in order to correct errors.

Let's describe the communication scenario more precisely and introduce some notation. After the communication phase, Alice and Bob hold two strings that we model using random variables X_A and X_B respectively. Define S, the *syndrome*, as $S = X_A \oplus X_B$, where we used $\oplus$ to denote the bitwise parity: for example, $010 \oplus 110 = 100$. Note that S is different from zero if and only if there are errors, and moreover S has a 1 in any position where X_A and X_B differ. The goal of information reconciliation is for Bob to correct his string X_B and recover X_A. The information reconciliation protocols we consider

are entirely classical and exchange all their messages over the CAC. For any such protocol we let C be the string consisting of all the messages exchanged during the protocol. Let $\hat{X}_A$ be Bob's final output, that he would like to equal X_A.

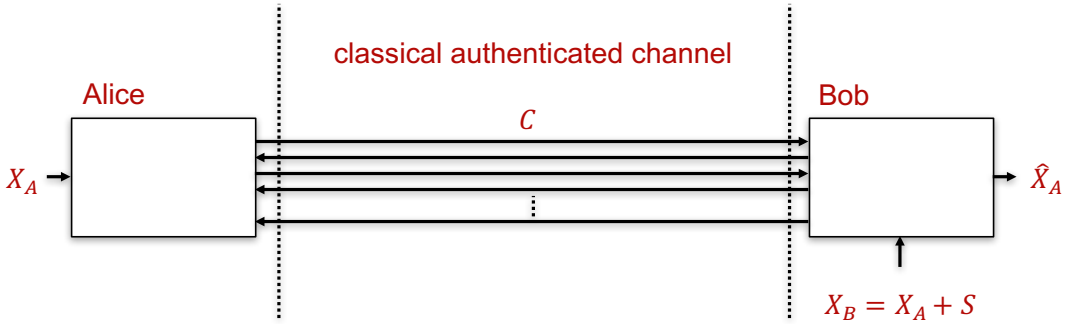

Fig. 6.2 Scheme of a general information reconciliation protocol.

Definition 6.4.1 (Information Reconciliation) *Let X_A and 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 :*

- $\text{Prob}(\hat{X}_A \neq X_A) \leq \varepsilon$.
- *The length of the messages exchanged on the CAC is $|C| \leq c$.*

Observe that in the definition we don't require that $\hat{X}_A = X_A$ with certainty. Although we might have liked to, similarly to the correctness requirement for QKD this would have been too strong a condition in general: there's always a chance that things go wrong, and our goal is to design protocols so that this chance is as small as possible. The reason for the second requirement is because any communication exchanged over the CAC leaks to the eavesdropper, Eve. Because in general we don't have a good way to control how this information is related to Alice and Bob's strings X_A and X_B we take a worst-case approach: worst-case, any bit exchanged during the protocol is a bit leaked about X_A. We will then apply the chain rule for the min-entropy (Exercise **??**) as

$$H_{\min}(X|EC) \geq H_{\min}(X|E) - |C| \,. \tag{6.7}$$

This equation bounds how much information is lost to Eve during information reconciliation. It is important to estimate how much uncertainty remains in Alice and Bob's strings before they perform privacy amplification as described in the previous section.

If we separate the two goals from Definition 6.4.1 then it is not hard to achieve them. Why is this? Imagine that a reconciliation protocol consists of Alice sending her whole string to Bob over the CAC. This is a great protocol if we only care about correctness, but the leakage $|C| = |X_A|$ is maximal and by (6.7) 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 at all, unless $\varepsilon = 1$.

Information reconciliation protocols can be classified depending on their use of the CAC. The most general protocol consists in the exchange of messages in both directions, from Alice to Bob and from Bob to Alice. We call such a protocol a *two-way* or an *interactive* protocol. However, much simpler, and

as it turns out often sufficient, protocols would consist of a single message from Alice to Bob. We refer to such protocols as *one-way* reconciliation protocols. Let's describe such a protocol in the next section.

6.4.1 Syndrome coding

The scheme that we will describe is based on error-correcting codes. Let us first review the definition and main elements of linear error-correcting codes, which are the most widely studied (and used) kind of code. To understand the formal definition below, recall that a finite field with q elements is a finite set $\mathbb{F}_q$ of size q equipped with addition and multiplication laws that satisfy certain natural requirements. For example, the real numbers $\mathbb{R}$ with addition and multiplication are a field, but it is not finite. For $q = 2$ we have $\mathbb{F}_2 = \{0, 1\}$ with addition modulo 2 and standard multiplication, and this is in fact the only finite field with two elements. If you have never seen finite fields before, it is simpler, and sufficient for our purposes, to take $q = 2$ in the following definition. Then $\mathbb{F}_2^n$ is simply the vector space of all n-dimensional vectors with entries in $\{0, 1\}$, with addition of vectors being done coordinate-wise modulo 2, e.g. $(001) + (101) = (100)$. With these preliminaries in place, let's give the definition of a linear error-correcting code.

Definition 6.4.2 (Linear code) *Let $\mathbb{F}_q$ be a finite field of size q and $1 \leq k \leq n$ two integers. An (n, k) q-ary linear code C is a linear subspace of $\mathbb{F}_q^n$ of dimension k. n is called the* length *of the code and k its* dimension.

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

Since a code is just a subspace, to define it we only need a way to characterize a subspace of $\mathbb{F}_2^n$. It is convenient to do this by using an $m \times n$-dimensional *parity check matrix* H with entries in $\mathbb{F}_2$, which specifies equations that every codeword should satisfy. The code is thus defined as the set of vectors v such that $H \cdot v = 0$, which are also referred to as *codewords*. The dimension of the code induced by H is the dimension of the kernel of H, which by linear algebra is $k = n - \text{rank}(H)$. Therefore, if we take the rows of H to be linearly independent then $k = n - m$ and so $m = n - k$.

The map $s_H : \mathbb{F}_2^n \to \mathbb{F}_2^m$ given by $v \mapsto H \cdot v$ is called the *syndrome map*. The goal of an error-correcting code is to encode information in such a way that it is resistant to errors, and the syndrome map is used to correct errors when they arise. Intuitively this map is useful because it detects when a vector is a valid codeword: if v is in the code then by definition $H \cdot v = 0$. More generally, if $s_H(v) \neq 0$ then the value $s_H(v)$ should give us information about errors, i.e. how to modify v into $w = v + e$ so that the modified vector w is in the code, $s_H(w) = 0$, and e can be interpreted as a small error.

Example 6.4.1 *Let $v = (011)^T$ and $H = \begin{pmatrix} 110 \\ 011 \end{pmatrix}$. Then $s_H(v) = H \cdot v = (10)^T$. This is not the zero vector, and so v is not a codeword. However, $w = v + (100)^T = (111)^T$ satisfies $s_H(v) = 0$ and so it is a valid codeword. In this case, can you write down all vectors in the code?* [Hint: to make sure that you did not forget anyone, count dimensions.] ■

Let's now show how to use any linear error-correcting code to construct a one-way information reconciliation protocol. The idea is very simple. First, Alice computes the syndrome $C_A = s_H(X_A)$ and sends it to Bob.

Let us now consider Bob's actions. We think of Bob's string X_B as a "noisy" version of X_A. His goal is to "decode", i.e. he wants to estimate the error string S such that $X_B = X_A \oplus S$ and then

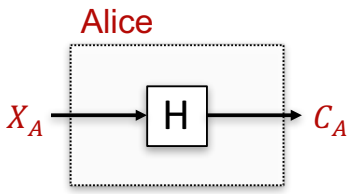

Fig. 6.3 The encoder in syndrome coding based reconciliation.

recover $X_A = X_B \oplus S$. Decoding is a little bit more complicated. The first step is that Bob computes the syndrome of X_B, that we call $C_B = s_H(X_B)$. Then Bob computes $C_S = C_B \oplus C_A$. This is the syndrome of the error string, i.e.

$$C_S = s_H(X_A) \oplus s_H(x_B) = s_H(X_A \oplus X_B) = s_H(S)$$

because matrix-vector multiplication is linear. Then, C_S is used by a sub-procedure, which depends on the choice of error-correcting code, that estimates the error string S from C_S and outputs the estimate. Call the estimate returned $\hat{S}$. In general it will always be the case that $\hat{S}$ is such that $s_H(\hat{S}) = C_S$. However, unless $m = n$ there will be many such strings (precisely, 2^{n-m}) and so we won't always have $\hat{S} = S$. For a good error-correcting code, this will be the case as long as S is "small enough", i.e. it has a small Hamming weight. How small is small enough is a property of the code called its "distance". Having recovered the estimate $\hat{S}$, Bob adds it to X_B to obtain $\hat{X}_B = X_B \oplus \hat{S}$ and this is his final output.

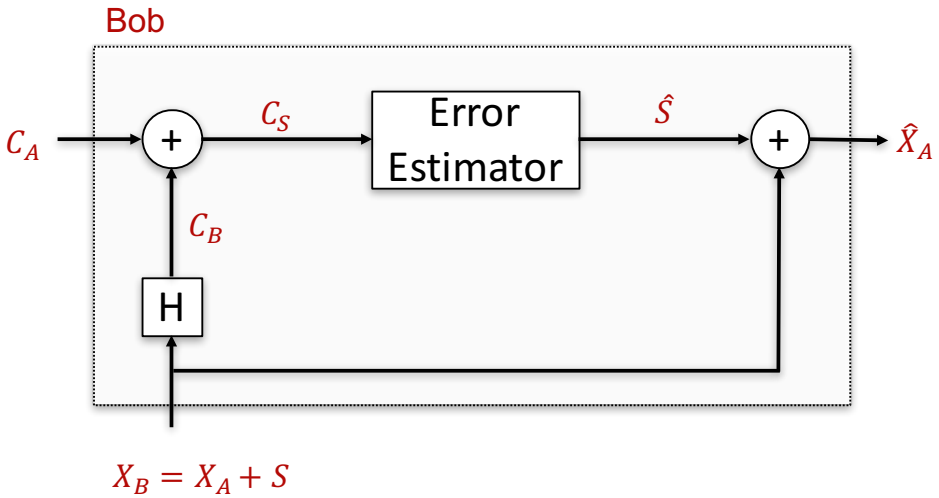

Fig. 6.4 The decoder in syndrome coding based reconciliation.

Example 6.4.2 *Let us go back to Example 6.4.1. We now describe the decoding. There are four*

different syndromes. We design our estimator function as follows:

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 are zero or one errors, the estimator will output the correct error estimate. In other words, among all possible error vectors with a given syndrome we always chose the one that has the smallest number of actual errors, i.e. the smallest possible number of 1. ∎

6.4.2 Some parameters

Now that we described the main idea for information reconciliation based on error-correcting codes, let's get a sense of the parameters that can be achieved, so that we can use such schemes in our QKD protocols in the next chapter.

In order to choose an information reconciliation protocol we need to have an idea of the error model, that is how the strings X_A and X_B relate to each other. A natural model is that each symbol of the n-bit strings X_A and X_B is drawn independently from the same joint distribution P_{AB}, where here A, B are binary random variables. In this case we can measure how much X_A and X_B typically differ using the conditional entropy

$$H(A|B) = H(A,B) - H(B) \ ,$$

where H is the Shannon entropy. If we further assume that A is uniformly distributed, and B is such that $B = A$ with probability $p = 1 - \delta$ and $B = 1 - A$ otherwise, then by direct calculation we find that $H(A|B) = h(\delta)$ where h is the binary entropy function. In this situation the theory of error-correcting codes tells us that as long as

$$m \ \geq \ n \cdot h(\delta) \tag{6.8}$$

then there exists an error-correcting code with parity-check matrix H and syndromes of size m such that for (X_A, X_B) distributed according to P_{AB}^n, with very high probability the string $\hat{S}$ of smallest Hamming weight such that $s_H(\hat{S}) = s_H(X_A \oplus X_B)$ will be precisely $S = X_A \oplus X_B$.[2] This means that in the protocol described in the previous section, if Bob computes $\hat{S}$ to be the smallest weight string whose syndrome is C_S, then he will recover $\hat{X}_A = X_A$ with high probability. In fact, for this to be the case it is sufficient to have the guarantee that $|S| \leq \delta n$ with high probability, i.e. we only need to know that the strings X_A and X_B do not differ too much (it is not necessary to know that each entry of the string has been generated independently according to the same distribution). Finally, it is also known that (6.8) is optimal in the sense that if we take m to be even slightly smaller than this, then in general there will be many possible error strings $\hat{S}$ of small weight that are compatible with a given syndrome C_S, and so it will not be possible to determine which $\hat{S}$ is the correct $S = X_A \oplus X_B$.

In practice it is not enough that there "exists" an error-correcting code with the right properties. First of all we need to know what that code is, i.e. what is the matrix H, and second we also need for there to be an efficient way to perform "syndrome decoding", i.e. recover $\hat{S}$ from C_S. For our purposes we will simply say that such codes do exist, and so whenever we want to perform information reconciliation we will be able to claim that it is possible to do so using leakage that scales as in (6.8), where the parameter δ will

[2] The probability that this is the case can be made very close to 1 by choosing m just a little bit larger than the lower bound in (6.8), e.g. $m \approx n \cdot h(\delta) + \Omega(\log(1/\delta))$ to get a probability $1 - \delta$.

be estimated in the protocol. Constructing such codes is by no means easy and it is a major achievement of the theory of error-correcting codes. We refer you to the chapter notes for bibliographical references.

6.5 Everlasting security

To end the chapter let us return to an important assumption that is always made when considering quantum key distribution protocols: that the communication channel between Alice and Bob, that we've been calling the "CAC", is what its name implies — an authenticated channel. This assumption is used to guarantee that, although the eavesdropper may intercept any communication, she cannot "impersonate" Alice or Bob by sending fake messages on the channel. If we allowed this, it could have dramatic consequences on the correctness of the protocol: for example, Eve could modify the syndrome information sent from Alice to Bob in information reconciliation and thereby make Bob think that he recovered $\hat{X}_A = X_A$ when this is absolutely not the case. Once we see concrete protocols in the next chapter you will see that if Eve could break the authentication she could make even much more devastating attacks.

So we do need a CAC. How reasonable is it to assume that we have it? This question is not only relevant for QKD; in general the access to an authenticated channel is a prerequisite for a large variety of cryptographic tasks. If we keep in mind that our goal in quantum cryptography is to implement protocols that have the fewest possible assumptions, we owe it to ourselves to examine this one critically. Indeed, there is no method to implement a CAC "out of the blue", unless one already has a shared secret; since this is the goal of QKD in the first place we certainly don't want to do that. Without assuming a prior secret, it is possible to show that constructing a CAC *requires* the use of computational assumptions on the power of the eavesdropper. It is beyond the scope of this book to explain how exactly authentication can be implemented, so we will simply reveal that the main primitive used to achieve this is called a "digital signature", and it can be implemented based on any trapdoor one-way function (such as the famous RSA function, whose security rests on the hardness of factoring—but of course with quantum adversaries you wouldn't want to rely on such an assumption!).

Is it reasonable to make computational assumptions, when one of the main goals of quantum key distribution is to provide information-theoretic security? This requirement is sometimes raised as a criticism against QKD, whose purpose is precisely to enable the secure exchange of a secret key *without* making computational assumptions. A key argument to counter this criticism and justify the use of computational assumptions for the CAC is the property of "everlasting security". As we will see in the next chapter, the key generated in a QKD protocol is secure as long as the CAC remains authenticated *for the duration of the protocol*. During this time it is indeed crucial that Eve is not able to send fake messages. However, once the protocol has ended and Alice and Bob have generated their private key, it is no longer relevant whether the channel remains authenticated or not. So the computational assumption guaranteeing security of the authenticated channel only needs to hold for a few seconds, and the key generated in the protocol will remain forever secure: Eve has no information about it, and will not be able to gain any additional information by breaking a channel that is no longer in use. This property distinguishes quantum key distribution from a much more naive protocol that would use standard cryptographic techniques to directly exchange the secret key; for any such method, if the authentication or any other cryptographic assumption used in the protocol is broken a year later, then the generated key is immediately made vulnerable.

6.6 Chapter notes

The formal security definition for quantum key distribution appears in the Ph.D. thesis of Renato Renner [**?**]. For a thorough discussion of the definition, including why it is "universally composable", we refer to [**?**].

Previous to the idea of using linear error correcting codes and one-way reconciliation there existed ad-hoc two-way protocols proposed specifically for the task of information reconciliation. The most well-known such protocol is Cascade [**?**], which has a reasonably simple description and has the advantage of being easy to implement.

The inequality (6.8) that quantifies the optimal leakage for any one-way information reconciliation protocol is due to Slepian and Wolf [**?**]. While the bound stated in the equality can be achieved in the limit $n \to \infty$, in practice one has to deal with finite values of n, and moreover efficiency considerations in terms of the decoding operation play an important role in the practicality of a protocol. Therefore it is common to accept a leakage that is marginally larger, of the form $\xi \cdot nH(X_A|X_B)$ where $\xi > 1$ is called the reconciliation efficiency. The constant ξ is often chosen $\xi \approx 1.2$ and this allows very efficient implementations, see [**?**] for more discussion.

In general it is an important open problem if there are two-way information reconciliation protocols that can achieve an asymptotically smaller leakage than one-way protocols. As we will see in the next chapter, in the context of quantum key distribution protocols any bit leaked for information reconciliation is a bit of key lost, and so minimizing this leakage is crucial to get optimized protocols.

For an introduction to the problem of authentication and how to solve it using private or public-key cryptography, see Chapter 5 in the notes [**?**].