Chapter 22

Digital Signatures Based on Discrete Logarithms

Public key signatures and their security notions were defined in Section 1.3.2. They are arguably the most important topic in public key cryptography (for example, to provide authentication of automatic software updates; see Section 1.1). This chapter gives some digital signature schemes based on the discrete logarithm problem. The literature on this topic is enormous and we only give a very brief summary of the area. RSA signatures are discussed in Section 24.6.

22.1 Schnorr Signatures

We assume throughout this section that an algebraic group G and an element $g \in G$ of prime order r are known to all users. The values (G, g, r) are known as **system parameters**. Let $h = g^a$ be a user's public key. A digital signature, on a message m with respect to a public key h, can be generated by a user who knows the private key a. It should be hard to compute a signature for a given public key without knowing the private key.

To explain the Schnorr signature scheme it is simpler to first discuss an identification scheme.

22.1.1 The Schnorr Identification Scheme

Informally, a **public key identification scheme** is a protocol between a Prover and a Verifier, where the Prover has a public key pk and private key sk, and the Verifier has a copy of pk. The protocol has three communication stages: first the Prover sends a commitment s_0; then the Verifier sends a challenge s_1; then the Prover sends a response s_2.

The Verifier either accepts or rejects the proof. The protocol is supposed to convince the Verifier that they are communicating with a user who knows the private key corresponding to the Prover's public key. In other words, the Verifier should be convinced that they are communicating with the Prover.

For the Schnorr scheme [522, 523] the Prover has public key $h = g^a$ where g is an element of an algebraic group of prime order r and $1 \leq a < r$ is chosen uniformly at random. The Prover chooses a random integer $0 \leq k < r$, computes $\mathsf{s}_0 = g^k$ and sends s_0 to the Verifier. The Verifier sends a "challenge" $1 \leq \mathsf{s}_1 < r$ to the Prover. The Prover returns $\mathsf{s}_2 = k + a\mathsf{s}_1 \pmod r$. The Verifier then checks whether

$$g^{\mathsf{s}_2} = \mathsf{s}_0 h^{\mathsf{s}_1} \tag{22.1}$$

and accepts the proof if this is the case. In other words, the Prover has successfully identified themself to the Verifier if the Verifier accepts the proof.

Exercise 22.1.1. Show that the Verifier in an execution of the Schnorr identification scheme does accept the proof when the Prover follows the steps correctly.

Exercise 22.1.2. Let $p = 311$ and $r = 31 \mid (p-1)$. Let $g = 169$, which has order r. Let $a = 11$ and $h = g^a \equiv 47 \pmod p$. Which of the following is a transcript $(\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2)$ of a correctly performed execution of the Schnorr identification scheme?

$$(15, 10, 12),\ (15, 10, 27),\ (16, 10, 12),\ (15, 16, 0).$$

Security of the Private Key

Unlike public key encryption (at least, under passive attacks), with identification schemes and digital signature schemes a user is always outputting the results of computations involving their private key. Hence, it is necessary to ensure that we do not leak information about the private key. An attack of this type on GGH signatures was given by Nguyen and Regev; see Section 19.11. Hence, we now explain why executions of the Schnorr identification protocol do not leak the private key.

A protocol (involving a secret) that does not leak any information about the secret is known as "zero knowledge". It is beyond the scope of this book to discuss this topic in detail, but we make a few remarks in the setting of the Schnorr identification scheme. First, consider a Verifier who really does choose s_1 independently and uniformly at random (rather than as a function of s_0 and h). It is easy to see that anyone can produce triples $(\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2)$ that satisfy equation (22.1), without knowing the private key (just choose s_1 and s_2 first and solve for s_0). Hence, a protocol transcript $(\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2)$ itself carries no information about a (this shows that the protocol is "honest verifier zero knowledge"). However, this argument does not imply that the protocol leaks no information about the secret to an adversary who chooses s_1 carefully. We now argue that the protocol is secure in this setting. The idea is to consider any pair $(\mathsf{s}_1, \mathsf{s}_2)$. Then, for every $1 \leq a < r$, there is some integer $0 \leq k < r$ such that $\mathsf{s}_2 \equiv k + a\mathsf{s}_1 \pmod r$. Now, if k were known to the verifier then they could solve for a. But, since the discrete logarithm problem is hard, it is computationally infeasible to determine any significant information about the distribution of k from s_0. Hence s_2 leaks essentially no information about a. Furthermore, there are no choices for s_1 that more readily allow the Verifier to determine a.

For security, k must be chosen uniformly at random; see Exercise 22.1.3 and Section 22.3 for attacks if some information on k is known. We stress that such attacks are much stronger than the analogous attacks for Elgamal encryption (see Exercise 20.4.1); there the adversary only learns something about a single message, whereas here they learn the private key!

Exercise 22.1.3. Suppose the random values k used by a prover are generated using the linear congruential generator $k_{i+1} = Ak_i + B \pmod{r}$ for some $1 \leq A, B < r$. Suppose an adversary knows A and B and sees two protocol transcripts $(\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2)$ and $(\mathsf{s}'_0, \mathsf{s}'_1, \mathsf{s}'_2)$ generated using consecutive outputs k_i and k_{i+1} of the generator. Show how the adversary can determine the private key a.

A generalisation of Exercise 22.1.3, where the modulus for the linear congruential generator is not r, is given by Bellare, Goldwasser and Micciancio [34].

Security Against Impersonation

Now we explain why the Verifier is convinced that the prover must know the private key a. The main ideas will also be used in the security proof of Schnorr signatures, so we go through the argument in some detail. First, we define an adversary against an identification protocol.

Definition 22.1.4. An **adversary against an identification protocol** (with an honest verifier) is a polynomial-time randomised algorithm A that takes as input a public key, plays the role of the Prover in the protocol with an honest Verifier, and tries to make the Verifier accept the proof. The adversary repeatedly and adaptively sends a value s_0, receives a challenge s_1 and answers with s_2 (indeed, the sessions of the protocol can be interleaved). The adversary is successful if the Verifier accepts the proof with noticeable probability (i.e., the probability, over all outputs s_0 by A and all choices for s_1, that the adversary can successfully respond with s_2 is at least one over a polynomial function of the security parameter). The protocol is secure if there is no successful adversary.

An adversary is just an algorithm A so it is reasonable to assume that A can be run in very controlled conditions. In particular, we will assume throughout this section that A can be repeatedly run so that it always outputs the same first commitment s_0 (think of A as a computer programme that calls a function `Random` to obtain random bits and then simply arrange that the function always returns the same values to A). This will allow us to respond to the same commitment with various different challenges s_1. Such an attack is sometimes known as a **rewinding attack** (Pointcheval and Stern [483] call it the **oracle replay attack**): If A outputs s_0, receives a challenge s_1, and answers with s_2 then re-running A on challenge s'_1 is the same as "rewinding" the clock back to when A had just output s_0 and then giving it a different challenge s'_1.

Theorem 22.1.5. *The Schnorr identification scheme is secure against impersonation (in the sense of Definition 22.1.4) if the discrete logarithm problem is hard.*

We first prove the result for perfect adversaries (namely, those that impersonate the user successfully every time the protocol is run). Later we discuss the result for more general adversaries.

Proof: (In the case of a perfect adversary) We build an expected polynomial-time algorithm (called the simulator) that solves a DLP instance (g, h) where g has prime order r and $h = g^a$ where $0 \leq a < r$ is chosen uniformly at random.

The simulator will play the role of the Verifier and will try to solve the DLP by interacting with A. First, the simulator starts A by giving it h as the public key and giving some choice for the function `Random`. The adversary outputs a value s_0, receives a response s_1 (chosen uniformly at random) from the simulator, then outputs s_2. Since A is perfect we assume that $(\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2)$ satisfy the verification equation.

First note that if values s_0 and s_2 satisfy equation (22.1) then s_0 lies in the group generated by g and so is of the form $\mathsf{s}_0 = g^k$ for some $0 \leq k < r$. Furthermore, it then follows that $\mathsf{s}_2 \equiv k + a\mathsf{s}_1 \pmod{r}$.

Now the simulator can re-run A on the same h and the same function `Random` (this is the rewinding). It follows that A will output s_0 again. The simulator then gives A a challenge $\mathsf{s}'_1 \neq \mathsf{s}_1$. Since A is perfect, it responds with s'_2 satisfying equation (22.1).

We have $\mathsf{s}_2 \equiv k + a\mathsf{s}_1 \pmod{r}$ and $\mathsf{s}'_2 \equiv k + a\mathsf{s}'_1 \pmod{r}$. Hence the simulator can compute $a \equiv (\mathsf{s}_2 - \mathsf{s}'_2)(\mathsf{s}_1 - \mathsf{s}'_1)^{-1} \pmod{r}$ and solve the DLP. In other words, if there is no polynomial-time algorithm for the DLP then there can be no polynomial-time adversary A against the protocol. □

The above proof gives the basic idea, but is not sufficient since we must consider adversaries that succeed with rather small probability. There are various issues to deal with. For example, A may not necessarily succeed on the first execution of the identification protocol. Hence, one must consider many executions $(\mathsf{s}_{i,0}, \mathsf{s}_{i,1}, \mathsf{s}_{i,2})$ for $1 \leq i \leq t$ and guess the value i into which one introduces the challenge $\mathsf{s}'_{i,1}$. Also, A may only succeed for a small proportion of the challenges s_1 for a given s_0 (it is necessary for the proof that A can succeed on two different choices of s_1 for the same value s_0). This latter issue is not a problem (since A succeeds with noticeable probability, it must succeed for a noticeable proportion of values s_1 for most values s_0). The former issue is more subtle and is solved using the forking lemma.

The **forking lemma** was introduced by Pointcheval and Stern [483]. A convenient generalisation has been given by Bellare and Neven [36]. The forking lemma determines the probability that a rewinding attack is successful. More precisely, consider an algorithm A (the adversary against the signature scheme) that takes as input a `Random` function and a list of responses s_1 to its outputs s_0. We will repeatedly choose a `Random` function and run A twice, the first time with a set of values $(\mathsf{s}_{1,1}, \ldots, \mathsf{s}_{t,1})$ being the responses in the protocol and the second time with a set $(\mathsf{s}_{1,1}, \ldots, \mathsf{s}_{j-1,1}, \mathsf{s}'_{j,1}, \ldots, \mathsf{s}'_{t,1})$ of responses for some $1 \leq j \leq t$. Note that A will output the same values $\mathsf{s}_{i,0}$ in both runs when $1 \leq i \leq j$. The forking lemma gives a lower bound on the probability that A succeeds in the identification protocol in the j-th execution as desired. Lemma 1 of [36] states that the success probability is at least $p(p/t - 1/r)$ where p is the success probability of A, t is the number of executions of the protocol in each game, and r is the size of the set of possible responses. Hence, if p is noticeable, t is polynomial and $1/r$ is negligible then the simulator solves the DLP with noticeable probability. We refer to [36, 483] and Section 10.4.1 of Vaudenay [616] for further details.

Exercise 22.1.6. Show that if the challenge values s_1 chosen by a Verifier can be predicted (e.g., because the Verifier is using a weak pseudorandom number generator) then a malicious player can impersonate an honest user in the Schnorr identification scheme.

Exercise 22.1.7. In the Schnorr identification scheme as presented above, the challenge is a randomly chosen integer $1 \leq \mathsf{s}_1 < r$. Instead, for efficiency[1] reasons, one could choose $1 \leq \mathsf{s}_1 < 2^l$ for some l such that $l \geq \kappa$ (where κ is the security parameter, but where 2^l is significantly smaller than r). Show that the proof of Theorem 22.1.5 still holds in this setting.

Exercise 22.1.8. Explain why the Schnorr identification scheme cannot be implemented in an algebraic group quotient.

22.1.2 Schnorr Signatures

We now present the **Schnorr signature scheme** [522, 523], which has very attractive security and efficiency. The main idea is to make the identification protocol of the previous

[1] One could speed up signature verification using similar methods to Exercise 22.1.13.

section non-interactive by replacing the challenge s_1 by a random integer that depends on the message being signed. This idea is known as the **Fiat-Shamir transform**. By Exercise 22.1.6 it is important that s_1 cannot be predicted and so it is also necessary to make it depend on s_0.

More precisely, one sets $\mathsf{s}_1 = H(\mathsf{m}\|\mathsf{s}_0)$ where H is a cryptographic hash function from $\{0,1\}^*$ to $\{0,1\}^l$ for some parameter l and where m and s_0 are interpreted as binary strings (and where $\|$ denotes concatenation of binary strings as usual).

One would therefore obtain the following signature scheme, which we call **naive Schnorr signatures**: To sign a message m choose a random $0 \leq k < r$, compute $\mathsf{s}_0 = g^k$, $\mathsf{s}_1 = H(\mathsf{m}\|\mathsf{s}_0)$ and $\mathsf{s}_2 = k + a\mathsf{s}_1 \pmod r$, and send the signature $(\mathsf{s}_0, \mathsf{s}_2)$ together with m. A verifier, given $\mathsf{m}, (\mathsf{s}_0, \mathsf{s}_2)$ and the public key h, would compute $\mathsf{s}_1 = H(\mathsf{m}\|\mathsf{s}_0)$ and accept the signature if

$$g^{\mathsf{s}_2} = \mathsf{s}_0 h^{\mathsf{s}_1}. \tag{22.2}$$

Schnorr makes the further observation that instead of sending $(\mathsf{s}_0, \mathsf{s}_2)$ one could send $(\mathsf{s}_1, \mathsf{s}_2)$. This has major implications for the size of signatures. For example, g may be an element of order r in $\mathbb{F}_p^*$ (for example, with $r \approx 2^{256}$ and $p \approx 2^{3072}$). In this case, $\mathsf{s}_0 = g^k$ requires 3072 bits, s_2 requires 256 bits, and s_1 may require as little as 128 bits. In other words, signatures would have $3072 + 256 = 3328$ bits in the naive scheme, whereas Schnorr signatures only require $128 + 256 = 384$ bits.

We present the precise Schnorr signature scheme in Figure 22.1.

KeyGen: This is the same as classic textbook Elgamal encryption. It outputs an algebraic group, an element g of prime order r, a public key $h = g^a$ and a private key $1 \leq a < r$ where a is uniformly chosen.

Sign(g, a, m): Choose uniformly at random $0 \leq k < r$, compute $\mathsf{s}_0 = g^k$, $\mathsf{s}_1 = H(\mathsf{m}\|\mathsf{s}_0)$ and $\mathsf{s}_2 = k + a\mathsf{s}_1 \pmod r$, where the binary string s_1 is interpreted as an integer in the usual way. The signature is $(\mathsf{s}_1, \mathsf{s}_2)$.

Verify$(g, h, \mathsf{m}, (\mathsf{s}_1, \mathsf{s}_2))$: Ensure that h is a valid public key for the user in question then test whether

$$\mathsf{s}_1 = H(\mathsf{m}\|g^{\mathsf{s}_2} h^{-\mathsf{s}_1}).$$

Figure 22.1: Schnorr Signature Scheme.

Example 22.1.9. Let $p = 311$ and $r = 31 \mid (p-1)$. Let $g = 169$ which has order r. Let $a = 11$ and $h = g^a \equiv 47 \pmod p$.

To sign a message m (a binary string) let $k = 20$ and $\mathsf{s}_0 = g^k \equiv 225 \pmod p$. The binary expansion of s_0 is $(11100001)_2$. We must now compute $\mathsf{s}_1 = H(\mathsf{m}\|11100001)$. Since we don't want to get into the details of H, let's just suppose that the output length of H is 4 and that s_1 is the binary string 1001. Then s_1 corresponds to the integer 9. Finally, we compute $\mathsf{s}_2 = k + a\mathsf{s}_1 \equiv 20 + 11 \cdot 9 \equiv 26 \pmod r$. The signature is $(\mathsf{s}_1, \mathsf{s}_2) = (9 = (1001)_2, 26)$. To verify the signature one computes

$$g^{\mathsf{s}_2} h^{-\mathsf{s}_1} = 169^{26} 47^{-9} \equiv 225 \pmod p$$

and checks that $\mathsf{s}_1 = H(\mathsf{m}\|11100001)$.

Exercise 22.1.10. Show that the Verify algorithm does succeed when given a pair $(\mathsf{s}_1, \mathsf{s}_2)$ output by the Sign algorithm.

22.1.3 Security of Schnorr Signatures

The security of Schnorr signatures essentially follows from the same ideas as used in Theorem 22.1.5. In particular, the security depends on the discrete logarithm problem (rather than CDH or DDH as is the case for Elgamal encryption). However, since the challenge is now a function of the message m and s_0, the exact argument of Theorem 22.1.5 cannot be used directly.

One approach is to replace the hash function by a random oracle H (see Section 3.7). The simulator can then control the values of H and the proof of Theorem 22.1.5 can be adapted to work in this setting. A careful analysis of Schnorr signatures in the random oracle model, using this approach and the forking lemma, was given by Pointcheval and Stern [483]. We refer to Theorem 14 of their paper for a precise result in the case where the output of H is $(\mathbb{Z}/r\mathbb{Z})^*$. A proof is also given in Section 10.4.2 of Vaudenay [616]. An analysis of the case where the hash function H maps to $\{0,1\}^l$ where $l < \log_2(r)$ is given by Neven, Smart and Warinschi [453].

There is no known proof of the security of Schnorr signatures in the standard model (even under very strong assumptions about the hash function). Paillier and Vergnaud [476] give evidence that one cannot give a reduction, in the standard model, from signature forgery for Schnorr signatures (with H mapping to $\mathbb{Z}/r\mathbb{Z}$) to DLP. More precisely, they show that if there is a reduction of a certain type (which they call an algebraic reduction) in the standard model from signature forgery for Schnorr signatures to DLP, then there is an algorithm for the "one-more DLP". We refer to [476] for the details.

We now discuss some specific ways to attack the scheme:

1. Given a signature $(\mathsf{s}_1, \mathsf{s}_2)$ on message m, if one can find a message m' such that $H(\mathsf{m} \| g^{\mathsf{s}_2} h^{-\mathsf{s}_1}) = H(\mathsf{m}' \| g^{\mathsf{s}_2} h^{-\mathsf{s}_1})$, then one has a signature also for the message m'. This fact can be used to obtain an existential forgery under a chosen-message attack.

   While one expects to be able to find hash collisions after roughly $2^{l/2}$ computations of H (see Section 3.2), what is needed here is not a general hash collision. Instead, we need a collision of the form $H(\mathsf{m} \| R) = H(\mathsf{m}' \| R)$ where $R = g^{\mathsf{s}_2} h^{-\mathsf{s}_1}$ is *not known* until a signature $(\mathsf{s}_1, \mathsf{s}_2)$ on m has been obtained. Hence, the adversary must first output a message m, then get the signature $(\mathsf{s}_1, \mathsf{s}_2)$ on m, then find m' such that $H(\mathsf{m} \| R) = H(\mathsf{m}' \| R)$. This is called the random-prefix second-preimage problem in Definition 4.1 of [453]. When R is sufficiently large it seems that solving this problem is expected to require around 2^l computations of H.

2. There is a passive existential forgery attack on Schnorr signatures if one can compute pre-images of H of a certain form. Precisely, choose any $(\mathsf{s}_1, \mathsf{s}_2)$ (for example, if the output of H is highly non-uniform then choose s_1 to be a "very likely" output of H), compute $R = g^{\mathsf{s}_2} h^{-\mathsf{s}_1}$, then find a bitstring m such that $H(\mathsf{m} \| R) = \mathsf{s}_1$. This attack is prevented if the hash function is hard to invert.

Hence, given a security parameter κ (so that breaking the scheme is required to take more than 2^κ bit operations) one can implement the Schnorr signature scheme with $r \approx 2^{2\kappa}$ and $l = \kappa$. For example, taking $\kappa = 128$, $2^{255} < r < 2^{256}$ and $l = 128$ gives signatures of 384 bits.

Exercise 22.1.11. ★ Fix $g \in G$ of order r and $\mathsf{m} \in \{0,1\}^*$. Can a pair $(\mathsf{s}_1, \mathsf{s}_2)$ be a Schnorr signature on the same message m for two different public keys? Are there any security implications of this fact?

22.1.4 Efficiency Considerations for Schnorr Signatures

The Sign algorithm performs one exponentiation, one hash function evaluation, and one computation modulo r. The Verify algorithm performs a multi-exponentiation $g^{\mathsf{s}_2}h^{-\mathsf{s}_1}$ where $0 \leq \mathsf{s}_2 < r$ and $1 \leq \mathsf{s}_1 < 2^l$ and one hash function evaluation. Hence, signing is faster than verifying.

There are a number of different avenues to speed up signature verification, depending on whether g is fixed for all users, whether one is always verifying signatures with respect to the same public key h or whether h varies, etc. We give a typical optimisation in Example 22.1.13. More dramatic efficiency improvements are provided by online/offline signatures (see Section 22.4), server-aided signature verification etc.

Exercise 22.1.12. Show how to modify the Schnorr signature scheme (with no loss of security) so that the verification equation becomes

$$\mathsf{s}_1 = H(\mathsf{m} \| g^{\mathsf{s}_2} h^{\mathsf{s}_1}).$$

Example 22.1.13. Suppose a server must verify many Schnorr signatures (using the variant of Exercise 22.1.12), always for the same value of g but for varying values of h. Suppose that $2^{l-1} < \sqrt{r} < 2^l$ (where l is typically also the output length of the hash function). One strategy to speed up signature verification is for the server to precompute and store the group element $g_1 = g^{2^l}$.

Given a signature $(\mathsf{s}_1, \mathsf{s}_2)$ with $0 \leq \mathsf{s}_1 < 2^l$ and $0 \leq \mathsf{s}_2 < r$ one can write $\mathsf{s}_2 = \mathsf{s}_{2,0} + 2^l \mathsf{s}_{2,1}$ with $0 \leq \mathsf{s}_{2,0}, \mathsf{s}_{2,1} < 2^l$. The computation of $g^{\mathsf{s}_2} h^{\mathsf{s}_1}$ is performed as the 3-dimensional multi-exponentiation (see Section 11.2)

$$g^{\mathsf{s}_{2,0}} g_1^{\mathsf{s}_{2,1}} h^{\mathsf{s}_1}.$$

The cost is roughly l squarings and $3l/2$ multiplications (the number of multiplications can be easily reduced using window methods, signed representations etc).

Schnorr [523] presents methods to produce the group elements g^k without having to perform a full exponentiation for each signature (the paper [523] is particularly concerned with making signatures efficient for smartcards). Schnorr's specific proposals were cryptanalysed by de Rooij [168].

22.2 Other Public Key Signature Schemes

The Schnorr signature scheme is probably the best public key signature scheme for practical applications.[2] A number of similar schemes have been discovered, the most well-known of which are Elgamal and DSA signatures. We discuss these schemes very briefly in this section.

22.2.1 Elgamal Signatures in Prime Order Subgroups

Elgamal [192] proposed the first efficient digital signature based on the discrete logarithm problem. We present the scheme for historical reasons, and because it gives rise to some nice exercises in cryptanalysis. For further details see Section 11.5.2 of [418] or Section 7.3 of [592].

[2] However, Schnorr signatures are not very widely used in practice. The reason for their lack of use may be the fact that they were patented by Schnorr.

Assume that g is an element of prime[3] order r in an algebraic group G. In this section we always think of G as being the "full" algebraic group (such as $\mathbb{F}_q^*$ or $E(\mathbb{F}_q)$) and assume that testing membership $g \in G$ is easy. The public key of user A is $h = g^a$ and the private key is a, where $1 \leq a < r$ is chosen uniformly at random.

The Elgamal scheme requires a function $F : G \to \mathbb{Z}/r\mathbb{Z}$. The only property required of this function is that the output distribution of F restricted to $\langle g \rangle$ should be close to uniform (in particular, F is not required to be hard to invert). In the case where $G = \mathbb{F}_p^*$ it is usual to define $F : \{0, 1, \ldots, p-1\} \to \{0, 1, \ldots, r-1\}$ by $F(n) = n \pmod r$. If G is the set of points on an elliptic curve over a finite field then one could define $F(x, y)$ by interpreting x (or x and y) as binary strings, letting n be the integer whose binary expansion is x (or $x \| y$), and then computing $n \pmod r$.

To sign a message m with hash $H(\mathsf{m}) \in \mathbb{Z}/r\mathbb{Z}$ one chooses a random integer $1 \leq k < r$, computes $\mathsf{s}_1 = g^k$, computes $\mathsf{s}_2 = k^{-1}(H(\mathsf{m}) - aF(\mathsf{s}_1)) \pmod r$, and returns $(\mathsf{s}_1, \mathsf{s}_2)$. To verify the signature $(\mathsf{s}_1, \mathsf{s}_2)$ on message m one checks whether $\mathsf{s}_1 \in \langle g \rangle$, $0 \leq \mathsf{s}_2 < r$, and

$$h^{F(\mathsf{s}_1)} \mathsf{s}_1^{\mathsf{s}_2} = g^{H(\mathsf{m})}$$

in G. Elgamal signatures are the same size as naive Schnorr signatures.

A striking feature of the scheme is the way that s_1 appears both as a group element and as an exponent (this is why we need the function F). In retrospect, this is a poor design choice for both efficiency and security. The following exercises explore these issues in further detail. Pointcheval and Stern give a variant of Elgamal signatures (the trick is to replace $H(\mathsf{m})$ by $H(\mathsf{m} \| \mathsf{s}_1)$) and prove the security in Sections 3.3.2 and 3.3.3 of [483].

Exercise 22.2.1. Show that the Verify algorithm succeeds if the Sign algorithm is run correctly.

Exercise 22.2.2. Show that one can verify Elgamal signatures by computing a single 3-dimensional multi-exponentiation. Show that the check $\mathsf{s}_1 \in \langle g \rangle$ can therefore be omitted if $\gcd(\mathsf{s}_2, \#G) = 1$. Hence, show that the time to verify an Elgamal signature when F and H map to $\mathbb{Z}/r\mathbb{Z}$ is around twice the time of the method in Example 22.1.13 to verify a Schnorr signature. Explain why choosing F and H to map to l-bit integers where $l \approx \log_2(r)/2$ does not lead to a verification algorithm as fast as the one in Example 22.1.13.

Exercise 22.2.3. (Elgamal [192]) Suppose the hash function H is deleted in Elgamal signatures (i.e., we are signing messages $\mathsf{m} \in \mathbb{Z}/r\mathbb{Z}$). Give a passive existential forgery in this case. (i.e., the attack only requires the public key).

Exercise 22.2.4. ★ Consider the Elgamal signature scheme in $\mathbb{F}_p^*$ with the function $F(n) = n \pmod r$. Suppose the function $F(n)$ computes $n \pmod r$ for all $n \in \mathbb{N}$ (not just $0 \leq n < p$) and that the check $\mathsf{s}_1 \in \langle g \rangle$ does not include any check on the size of the integer s_1 (for example, it could simply be the check that $\mathsf{s}_1^r \equiv 1 \pmod p$ or the implicit check of Exercise 22.2.2). Give a passive selective forgery attack.

Exercise 22.2.5. Consider the following variant of Elgamal signatures in a group $\langle g \rangle$ of order r: The signature on a message m for public key h is a pair $(\mathsf{s}_1, \mathsf{s}_2)$ such that $0 \leq \mathsf{s}_1, \mathsf{s}_2 < r$, and

$$h^{\mathsf{s}_1} g^{H(\mathsf{m})} = g^{\mathsf{s}_2}.$$

Show how to determine the private key of a user given a valid signature.

[3] The original Elgamal signature scheme specifies that g is a primitive root in $\mathbb{F}_p^*$, but for compatibility with all other cryptographic protocols in this book we have converted it to work with group elements of prime order in any algebraic group.

Exercise 22.2.6.★ (Bleichenbacher [66]) Consider the Elgamal encryption scheme in $\mathbb{F}_p^*$ with the function $F(n) = n \pmod r$. Suppose the checks $\mathsf{s}_1 \in \langle g \rangle$ and $0 \leq \mathsf{s}_2 < r$ are not performed by the Verify algorithm. Show how an adversary who has maliciously chosen the system parameter g can produce selective forgeries for any public key under a passive attack.

Exercise 22.2.7. (Vaudenay [615]) Let H be a hash function with l-bit output. Show how to efficiently compute an l-bit prime r, and messages $\mathsf{m}_1, \mathsf{m}_2$ such that $H(\mathsf{m}_1) \equiv H(\mathsf{m}_2) \pmod r$. Hence, show that if one can arrange for an algebraic group with subgroup of order r to be used as the system parameters for a signature scheme then one can obtain a signature on m_1 for any public key h by obtaining from user A a signature on m_2.

A convenient feature of Elgamal signatures is that one can verify a batch of signatures faster than individually verifying each of them. Some details are given in Exercise 22.2.8. Early work on this problem was done by Naccache, M'Raïhi, Vaudenay and Raphaeli [449] (in the context of DSA) and Yen and Laih [637]. Further discussion of the problem is given by Bellare, Garay and Rabin [33].

Exercise 22.2.8. Let $(\mathsf{s}_{1,i}, \mathsf{s}_{2,i})$ be purported signatures on messages m_i with respect to public keys h_i for $1 \leq i \leq t$. A verifier can choose random integers $1 \leq w_i < r$ and verify all signatures together by testing whether $\mathsf{s}_{1,i} \in \langle g \rangle$ and $0 \leq \mathsf{s}_{2,i} < r$ for all i and the single equation

$$\left(\prod_{i=1}^{t} h_i^{w_i F(\mathsf{s}_{1,i})} \right) \left(\prod_{i=1}^{t} \mathsf{s}_{1,i}^{w_i \mathsf{s}_{2,i}} \right) = g^{\sum_{i=1}^{t} w_i H(\mathsf{m}_i)}. \tag{22.3}$$

Show that if all the signatures $(\mathsf{s}_{1,i}, \mathsf{s}_{2,i})$ are valid then the batch is declared valid. Show that if there is at least one invalid signature in the batch then the probability the batch is declared valid is at most $1/(r-1)$. Show how to determine, with high probability, the invalid signatures using a binary search.

If one uses the methods of Exercise 22.2.2 then verifying the t signatures separately requires t three-dimensional multi-exponentiations. One can break equation (22.3) into about $2t/3$ three-dimensional multi-exponentiations. So, for groups where testing $\mathsf{s}_{1,i} \in \langle g \rangle$ is easy (e.g., elliptic curves of prime order), the batch is asymptotically verified in about $2/3$ the time of verifying the signatures individually. Show how to speed up verification of a batch of signatures further if the public keys h_i are all equal. How much faster is this than verifying the signatures individually?

Yen and Laih [637] consider batch verification of naive Schnorr signatures as mentioned in Section 22.1.2. Given t signatures $(\mathsf{s}_{0,i}, \mathsf{s}_{2,i})$ on messages m_i and for keys h_i, Yen and Laih choose $1 \leq w_i < 2^l$ (for a suitable small value of l; they suggest $l = 15$) and verify the batch by testing $\mathsf{s}_{0,i} \in \langle g \rangle$, $0 \leq \mathsf{s}_{2,i} < r$ and

$$g^{\sum_{i=1}^{t} w_i \mathsf{s}_2} = \prod_{i=1}^{t} \mathsf{s}_{0,i}^{w_i} \prod_{i=1}^{t} h_i^{w_i H(\mathsf{m}_i \| \mathsf{s}_{0,i})}.$$

Give the verification algorithm when the public keys are all equal. Show that the cost is roughly $l/(3\log_2(r))$ times the cost of verifying t Elgamal signatures individually.

Explain why it seems impossible to verify batches of Schnorr signatures faster than verifying each individually.

22.2.2 DSA

A slight variant of the Elgamal signature scheme was standardised by NIST[4] as a digital signature standard. This is often called **DSA**.[5] In the case where the group G is an elliptic curve then the scheme is often called **ECDSA**.

In brief, the scheme has the usual public key $h = g^a$ where g is an element of prime order r in an algebraic group G and $1 \leq a < r$ is chosen uniformly at random. As with Elgamal signatures, a function $F : G \to \mathbb{Z}/r\mathbb{Z}$ is required. To sign a message with hash value $H(\mathsf{m})$ one chooses a random $1 \leq k < r$ and computes $\mathsf{s}_1 = F(g^k)$. If $\mathsf{s}_1 = 0$ then repeat[6] for a different value of k. Then compute $\mathsf{s}_2 = k^{-1}(H(\mathsf{m}) + a\mathsf{s}_1) \pmod r$ and, if $\mathsf{s}_2 = 0$ then repeat for a different value of k. The signature on message m is $(\mathsf{s}_1, \mathsf{s}_2)$. To verify the signature one first checks that $1 \leq \mathsf{s}_1, \mathsf{s}_2 < r$, then computes $u_1 = H(\mathsf{m})\mathsf{s}_2^{-1} \pmod r$, $u_2 = \mathsf{s}_1\mathsf{s}_2^{-1} \pmod r$, then determines whether or not

$$\mathsf{s}_1 = F(g^{u_1}h^{u_2}). \tag{22.4}$$

Note that a DSA signature is a pair of integers modulo r so is shorter in general than an Elgamal signature but longer in general than a Schnorr signature. Verification is performed using multi-exponentiation.

Exercise 22.2.9. Show that Verify succeeds on values output by Sign.

Exercise 22.2.10. The case $\mathsf{s}_1 = 0$ is prohibited in DSA signatures. Show that if this check was omitted and if an adversary could find an integer k such that $F(g^k) = 0$ then the adversary could forge DSA signatures for any message. Hence show that, as in Exercise 22.2.6, an adversary who maliciously chooses the system parameters could forge signatures for any message and any public key.

Exercise 22.2.11. The case $\mathsf{s}_2 = 0$ is prohibited in DSA signatures since the Verify algorithm fails when s_2 is not invertible. Show that if a signer outputs a signature $(\mathsf{s}_1, \mathsf{s}_2)$ produced by the Sign algorithm but with $\mathsf{s}_2 = 0$ then an adversary would be able to determine the private key a.

We saw in Exercise 22.2.2 that verifying Elgamal signatures is slow compared with verifying Schnorr signatures using the method in Example 22.1.13. Exercise 22.2.12 shows a variant of DSA (analogous to naive Schnorr signatures) that allows signature verification closer in speed to Schnorr signatures.

Exercise 22.2.12. (Antipa, Brown, Gallant, Lambert, Struik and Vanstone [11]) Consider the following variant of the DSA signature scheme: To sign m choose $1 \leq k < r$ randomly, compute $\mathsf{s}_0 = g^k$, $\mathsf{s}_2 = k^{-1}(H(m) + aF(\mathsf{s}_0)) \pmod r$ and return $(\mathsf{s}_0, \mathsf{s}_2)$. To verify $(\mathsf{m}, \mathsf{s}_0, \mathsf{s}_2)$ one computes $u_1 = H(\mathsf{m})\mathsf{s}_2^{-1} \pmod r$, $u_2 = F(\mathsf{s}_0)\mathsf{s}_2^{-1} \pmod r$ as in standard DSA and checks whether

$$\mathsf{s}_0 = g^{u_1}h^{u_2}. \tag{22.5}$$

Show that one can also verify the signature by checking, for any $1 \leq v < r$, whether

$$\mathsf{s}_0^v = g^{u_1 v}h^{u_2 v}. \tag{22.6}$$

Show that one can efficiently compute an integer $1 \leq v < r$ such that the equation (22.6) can be verified more quickly than equation (22.5).

[4] NIST stands for "National Institute of Standards and Technology" and is an agency that develops technology standards for the USA.

[5] DSA stands for "digital signature algorithm".

[6] The events $\mathsf{s}_1 = 0$ and $\mathsf{s}_2 = 0$ occur with negligible probability and so do not effect the performance of the signing algorithm.

There is no proof of security for DSA signatures in the standard or random oracle model. A proof of security in the random oracle model of a slightly modified version of DSA (the change is to replace $H(\mathsf{m})$ with $H(\mathsf{m}\|\mathsf{s}_1)$) was given by Pointcheval and Vaudenay [484, 105] (also see Section 10.4.2 of [616]). A proof of security for DSA in the generic group model[7] was given by Brown; see Chapter II of [65].

Exercise 22.2.13. Consider a digital signature scheme where a signature on message m with respect to public key h is an integer $0 \leq \mathsf{s} < r$ such that

$$\mathsf{s} = H(\mathsf{m}\|h^{\mathsf{s}}).$$

What is the problem with this signature scheme?

22.2.3 Signatures Secure in the Standard Model

None of the signature schemes considered so far has a proof of security in the standard model. Indeed, as mentioned, Paillier and Vergnaud [476] give evidence that Schnorr signatures cannot be proven secure in the standard model. In this section we briefly mention a signature scheme due to Boneh and Boyen [76, 77] that is secure in the standard model. However, the security relies on a very different computational assumption than DLP and the scheme needs groups with an extra feature (namely, a pairing; see Definition 22.2.14). We present a simple version of their scheme that is unforgeable under a weak chosen-message attack if the q-strong Diffie-Hellman problem holds (these notions are defined below).

We briefly introduce pairing groups (more details are given in Chapter 26). We use multiplicative notation for pairing groups, despite the fact that G_1 and G_2 are typically subgroups of elliptic curves over finite fields and hence are usually written additively.

Definition 22.2.14. (**Pairing groups**) Let G_1, G_2, G_T be cyclic groups of prime order r. A **pairing** is a map $e : G_1 \times G_2 \to G_T$ such that

1. e is non-degenerate and bilinear, i.e., $g_1 \in G_1 - \{1\}$ and $g_2 \in G_2 - \{1\}$ implies $e(g_1, g_2) \neq 1$ and $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for $a, b \in \mathbb{Z}$,
2. there is a polynomial-time algorithm to compute $e(g_1, g_2)$.

For the Boneh-Boyen scheme we also need there to be an efficiently computable injective group homomorphism $\psi : G_2 \to G_1$ (for example, a distortion map; see Section 26.6.1).

We will assume that elements in G_1 have a compact representation (i.e., requiring not much more than $\log_2(r)$ bits) whereas elements of G_2 do not necessarily have a compact representation. The signature is an element of G_1 and hence is very short. Figure 22.2 gives the (weakly secure) Boneh-Boyen Signature Scheme.

Exercise 22.2.15. Show that if the Verify algorithm for weakly secure Boneh-Boyen signatures accepts (m, s) then $\mathsf{s} = g_1^{(\mathsf{m}+a)^{-1} \pmod r}$.

The Boneh-Boyen scheme is unforgeable under a weak chosen-message attack if the q-strong Diffie-Hellman problem holds. We define these terms now.

Definition 22.2.16. A **weak chosen-message attack** (called a **generic chosen-message attack** by Goldwasser, Micali and Rivest [258]) on a signature scheme is an

[7] The generic group model assumes that any algorithm to attack the scheme is a generic algorithm for the group G. This seems to be a reasonable assumption when using elliptic curves.

KeyGen: Choose $g_2 \in G_2 - \{1\}$ uniformly at random and set $g_1 = \psi(g_2)$. Let $z = e(g_1, g_2)$. Choose $1 \leq a < r$ and set $u = g_2^a$. The public key is $\mathsf{pk} = (g_1, g_2, u, z)$ and the private key is a.

Sign(m, a): We assume that $\mathsf{m} \in \mathbb{Z}/r\mathbb{Z}$. If $a + \mathsf{m} \equiv 0 \pmod r$ then return $\perp$, else compute $\mathsf{s} = g_1^{(a+\mathsf{m})^{-1} \pmod r}$ and return s.

Verify$(\mathsf{m}, \mathsf{s}, \mathsf{pk})$: Check that $\mathsf{s} \in G_1$ and then check that

$$e(\mathsf{s}, ug_2^{\mathsf{m}}) = z.$$

Figure 22.2: Weakly Secure Boneh-Boyen Signature Scheme.

adversary A that outputs a list $\mathsf{m}_1, \ldots, \mathsf{m}_t$ of messages, receives a public key and signatures $\mathsf{s}_1, \ldots, \mathsf{s}_t$ on these messages, and then must output a message m and a signature s. The adversary wins if Verify$(\mathsf{m}, \mathsf{s}, \mathsf{pk})$ = "valid" and if $\mathsf{m} \notin \{\mathsf{m}_1, \ldots, \mathsf{m}_t\}$.

Hence, a weak chosen message attack is closer to a known message attack than a chosen message attack.

Definition 22.2.17. Let $q \in \mathbb{N}$ (not necessarily prime). Let G_1, G_2, G_T be pairing groups as in Definition 22.2.14. Let $g_1 \in G_1 - \{1\}$ and $g_2 \in G_2 - \{1\}$. The **q-strong Diffie-Hellman problem** (q-**SDH**) is, given $(g_1, g_2, g_2^a, g_2^{a^2}, \ldots, g_2^{a^q})$, where $1 \leq a < r$ is chosen uniformly at random, to output a pair

$$\left(\mathsf{m}, g_1^{(\mathsf{m}+a)^{-1} \pmod r}\right)$$

where $0 \leq \mathsf{m} < r$.

This problem may look rather strange at first sight since the value q can vary. The problem is mainly of interest when q is polynomial in the security parameter (otherwise, reading the problem description is not polynomial-time). Problems (or assumptions) like this are sometimes called parameterised since there is a parameter (in this case q) that determines the size for a problem instance. Such problems are increasingly used in cryptography, though many researchers would prefer to have systems whose security relies on more familiar assumptions.

There is evidence that the computational problem is hard. Theorem 5.1 of Boneh and Boyen [76] shows that a generic algorithm for q-SDH needs to make $\Omega(\sqrt{r/q})$ group operations to have a good chance of success. The algorithms of Section 21.5 can be used to solve q-SDH. In particular, if $q \mid (r-1)$ (and assuming $q < \sqrt{r}$) then Theorem 21.5.1 gives an algorithm to compute a with complexity $O(\sqrt{r/q})$ group operations, which meets the lower bound for generic algorithms.

Exercise 22.2.18. Show that one can use ψ and e to verify that the input to an instance of the q-SDH is correctly formed. Similarly, show how to use e to verify that a solution to a q-SDH instance is correct.

Theorem 22.2.19. *If the q-SDH problem is hard then the weak Boneh-Boyen signature scheme is secure under a weak chosen message attack, where the adversary requests at most $q-1$ signatures.*

Proof: (Sketch) Let $(g_1, g_2, g_2^a, g_2^{a^2}, \ldots, g_2^{a^q})$ be a q-SDH instance and let A be an adversary against the scheme. Suppose A outputs messages $\mathsf{m}_1, \ldots, \mathsf{m}_t$ with $t < q$.

Without loss of generality, $t = q - 1$ (since one can just add dummy messages). The idea of the proof is to choose a public key so that one knows $g_1^{(\mathsf{m}_i+a)^{-1}}$ for all $1 \leq i \leq t$. The natural way to do this would be to set

$$g_1' = g_1^{\prod_{i=1}^{t}(\mathsf{m}_i+a)}$$

but the problem is that we don't know a. The trick is to note that $F(a) = \prod_{i=1}^{t}(\mathsf{m}_i+a) = \sum_{i=0}^{t} F_i a^i$ is a polynomial in a with explicitly computable coefficients in $\mathbb{Z}/r\mathbb{Z}$. One can therefore compute $g_2' = g_2^{F(a)}$, $g_1' = \psi(g_2')$ and $h = g_2^{aF(a)}$ using, for example,

$$g_2' = \prod_{i=0}^{t} \left(g^{a^i}\right)^{F_i}.$$

Similarly, one can compute signatures for all the messages m_i. Hence, the simulator provides to A the public key $(g_1', g_2', h, z' = e(g_1', g_2'))$ and all t signatures.

Eventually, A outputs a forgery (m, s) such that $\mathsf{m} \neq \mathsf{m}_i$ for $1 \leq i \leq t$. If $t < q - 1$ and q is polynomial in the security parameter then m is one of the dummy messages with negligible probability $(q - 1 - t)/r$. One has

$$\mathsf{s} = g_1'^{(\mathsf{m}+a)^{-1} \pmod r} = g_1^{F(a)(\mathsf{m}+a)^{-1} \pmod r}.$$

The final trick is to note that the polynomial $F(a)$ can be written as $G(a)(a + \mathsf{m}) + c$ for some explicitly computable values $G(a) \in (\mathbb{Z}/r\mathbb{Z})[a]$ and $c \in (\mathbb{Z}/r\mathbb{Z})^*$. Hence, the rational function $F(a)/(a + \mathsf{m})$ can be written as

$$\frac{F(a)}{a+\mathsf{m}} = G(a) + \frac{c}{a+\mathsf{m}}.$$

One can therefore deduce $g_1^{(a+\mathsf{m})^{-1} \pmod r}$ as required. □

A fully secure signature scheme is given in [76] and it requires an extra element in the public key and an extra element (in $\mathbb{Z}/r\mathbb{Z}$) in the signature. The security proof is rather more complicated, but the essential idea is the same.

Jao and Yoshida [313] showed the converse result, namely that if one can solve q-SDH then one can forge signatures for the Boneh-Boyen scheme.

22.3 Lattice Attacks on Signatures

As mentioned earlier, there is a possibility that signatures could leak information about the private key. Indeed, Nguyen and Regev [457] give such an attack on lattice-based signatures.

The aim of this section is to present an attack due to Howgrave-Graham and Smart [299]. They determine the private key when given some signatures and given some bits of the random values k (for example, due to a side-channel attack or a weak pseudorandom number generator). The analysis of their attack was improved by Nguyen and Shparlinski [459, 460] (also see Chapter 16 of [558]).

The attack works for any signature scheme where one can obtain from a valid signature a linear equation modulo r with two unknowns, namely the private key a and the randomness k. We now clarify that this attack applies to the s_2 value for the Schnorr, Elgamal and DSA signature schemes:

Schnorr: $\mathsf{s}_2 \equiv k + a\mathsf{s}_1 \pmod r$ where $\mathsf{s}_1, \mathsf{s}_2$ are known.

Elgamal: $k\mathsf{s}_2 \equiv H(\mathsf{m}) - aF(\mathsf{s}_1) \pmod{r}$ where $H(\mathsf{m}), F(\mathsf{s}_1)$ and s_2 are known.

DSA: $k\mathsf{s}_2 \equiv H(\mathsf{m}) + a\mathsf{s}_1 \pmod{r}$ where $H(\mathsf{m}), \mathsf{s}_1$ and s_2 are known.

Suppose we are given a message m and a valid signature $(\mathsf{s}_1, \mathsf{s}_2)$ and also the l most or least significant bits of the random value k used by the Sign algorithm to generate s_1. Writing these known bits as k_0 we have, in the case of least significant bits, $k = k_0 + 2^l z$ where $0 \leq z < r/2^l$. Indeed, one gets a better result by writing

$$k = k_0 + 2^l \lfloor r/2^{l+1} \rfloor + 2^l z \tag{22.7}$$

with $-r/2^{l+1} \leq z \leq r/2^{l+1}$. Then one can re-write any of the above linear equations in the form

$$z \equiv ta - u \pmod{r}$$

for some $t, u \in \mathbb{Z}$ that are known. In other words, we know

$$\Big(t, u = \mathrm{MSB}_l(at \pmod{r})\Big), \tag{22.8}$$

which is an instance of the hidden number problem (see Section 21.7.1).

If the values t in equation (22.8) are uniformly distributed in $\mathbb{Z}/r\mathbb{Z}$ then Corollary 21.7.10 directly implies that if $r > 2^{32}$ and if we can determine $l \geq \lceil \sqrt{\log_2(r)} \rceil + \lceil \log_2(\log_2(r)) \rceil$ consecutive bits of the randomness k then one can determine the private key a in polynomial-time given $n = 2\lceil \sqrt{\log_2(r)} \rceil$ message-signature pairs. As noted in [459], in the practical attack the values t arising are not uniformly distributed. We refer to [459, 460] for the full details. In practice, the attack works well for current values for r when $l = 4$, see Section 4.2 of [459].

Exercise 22.3.1. Show how to obtain an analogue of equation (22.7) when the l most significant bits are known.

Bleichenbacher has described a similar attack on a specific implementation of DSA that used a biased random generator for the values k.

22.4 Other Signature Functionalities

There are many topics that are beyond the scope of this book. We briefly mention some of them now.

- One time signatures. These are fundamental in provable security and are used as a tool in many theoretical public key schemes. However, since these are usually realised without using the number theoretic structures presented in this book we do not give the details. Instead, we refer the reader to Section 11.6 of [418], Section 12.5 of [334] and Section 7.5.1 of [592].

- Online/offline signatures. The goal here is to design public key signature schemes that possibly perform some (slow) precomputations when they are "offline" but that generate a signature on a given message m extremely quickly. The typical application is smart cards or other tokens that may have extremely constrained computing power.

  The first to suggest a solution to this problem seems to have been Schnorr in his paper [522] on efficient signatures for smart cards. The Schnorr signature scheme

already has this functionality: if $\mathsf{s}_0 = g^k$ is precomputed during the idle time of the device, then generating a signature on message m only requires computing $\mathsf{s}_1 = H(\mathsf{m}\|\mathsf{s}_0)$ and $\mathsf{s}_2 = k + a\mathsf{s}_1 \pmod r$. The computation of s_1 and s_2 is relatively fast since no group operations are performed.

A simple idea due to Girault [254] (proposed for groups of unknown order, typically $(\mathbb{Z}/N\mathbb{Z})^*$) is to make Schnorr signatures even faster by omitting the modular reduction in the computation of s_2. In other words, k, a, s_1 are all treated as integers and s_2 is computed as the integer $k + a\mathsf{s}_1$. To maintain security it is necessary to take k to be bigger than $2^l r$ (i.e., bigger than any possible value for the integer $a\mathsf{s}_1$). This idea was fully analysed (and generalised to groups of known order) by Girault, Poupard and Stern [255].

- Threshold signatures. The idea is to have a signature that can only be generated by a collection of users. There is a large literature on this problem and we do not attempt a full treatment of the subject here.

  A trivial example is when two users hold additive shares a_1, a_2 of a private key (in other words, $h = g^{a_1+a_2} = g^{a_1}g^{a_2}$ is the public key). A Schnorr signature on message m can be computed as follows: User $i \in \{1, 2\}$ chooses a random integer $0 \leq k_i < r$, computes g^{k_i}, and sends it to the other. Both users can compute $\mathsf{s}_0 = g^{k_1}g^{k_2}$. User $i \in \{1, 2\}$ can then compute $\mathsf{s}_1 = H(\mathsf{m}\|\mathsf{s}_0)$ and $\mathsf{s}_{2,i} = k_i + a_i\mathsf{s}_1 \pmod r$. The signature is $(\mathsf{s}_0, \mathsf{s}_{2,1} + \mathsf{s}_{2,2} \pmod r)$.

- Signatures with message recovery. Usually a signature and a message are sent together. Signatures with message recovery allow some (or all) of the message to be incorporated in the signature. The whole message is recovered as part of the signature verification process. We refer to Section 11.5.4 of [418] for Elgamal signatures with message recovery.

- Undeniable signatures. These are public key signatures that can only be verified by interacting with the signer (or with some other designated verifier). A better name would perhaps be "invisible signatures" or "unverifiable signatures". We refer to Section 7.6 of [592].

- Identity-Based Signatures. Identity-based cryptography is a concept introduced by Shamir. The main feature is that a user's public key is defined to be a function of their "identity" (for example, their email address) together with some master public key. Each user obtains their private key from a Key Generation Center that possesses the master secret. One application of identity-based cryptography is to simplify public-key infrastructures.

  An identity-based signature is a public-key signature scheme for which it is not necessary to verify a public key certificate on the signer's key before verifying the signature (though note that it may still be necessary to verify a certificate for the master key of the system). There are many proposals in the literature, but we do not discuss them in this section. One example is given in Section 24.6.3).