7

Password-Based Protocols

7.1 Introduction

Cryptographic authentication relies on possession of a key by the party to be authenticated. Such a key is usually chosen randomly within its domain and can be of lengths from around 100 bits up to many thousands of bits, depending on the algorithm used and security level desired. Experience has shown [109, 333] that humans find it difficult to remember secrets in the form of passwords of even seven or eight characters. But if all upper and lower case letters are used together with the digits 0 to 9 then a random eight-character password represents less than 48 bits of randomness. Therefore we can conclude that even short random keys for cryptographic algorithms cannot be reliably remembered by humans. Another way to express this is that it can be assumed that a computer is able to search through all possible passwords in a short time.

Cryptographic keys are often stored in secure memory in computers or using special devices such as tamper-resistant cryptographic servers or in smart cards. However, there are situations where this is inconvenient or expensive. Not all devices are tamper resistant, and the memory required for public keys can be scarce. Therefore it is desirable to be able to set up secure communications relying only on a short secret that can be remembered by humans.

This chapter examines a number of key establishment protocols that have been designed to be secure in the situation that the principals share only a password of small entropy. At first thought it might seem impossible to achieve key establishment using only a short secret in such a way that brute force searching to find the secret is not possible. This intuition may account for why it was not until 1989 that the first password-based protocols appeared in the literature. These first protocols, due to Lomas *et al.* [210], used the additional assumption that the client (in a client–server application) has knowledge of the server's public key, in addition to sharing the password with the server. Later Bellovin and Merritt [38] introduced a class of protocols that does not require this assumption.

C. Boyd et al., *Protocols for Authentication and Key Establishment*

The idea of Bellovin and Merritt's Encrypted Key Exchange (EKE) protocols is that the protocol initiator will choose an ephemeral public key and use the shared password to *encrypt* this key. The responder can decrypt the public key and use it to send the session key securely back to the initiator. On the assumption (not always reasonable) that public keys are random strings, an adversary who tries a brute force search of passwords will not be able to distinguish which ephemeral public key was used; furthermore even when the correct public key has been found it cannot be used to discover the session key since it is not possible to obtain the private key from the public key. It is interesting to compare password-based protocols with protocols providing forward secrecy. Both seem to be possible only through use of ephemeral public keys[1] and, for both, a typical approach is to use long-term keys (which may be passwords) only for authentication of the message exchanges. Many password-based protocols do provide forward secrecy.

We may summarise the special properties and requirements for password-based key establishment protocols as follows. Additional properties, such as forward secrecy, are often seen as desirable.

- Users only possess a secret of small entropy. Specifically, it is possible for the adversary to search through all possible secrets in a reasonable time.
- Off-line dictionary attacks should not be feasible. This means that a passive eavesdropper who records the transcript of one or more sessions cannot eliminate a significant number of possible passwords.
- On-line dictionary attacks should not be feasible. This means that an active adversary cannot abuse the protocol so as to eliminate a significant number of possible passwords. An active adversary can always eliminate at least one password per protocol run by attempting to masquerade using that password. Ideally this should be all that the adversary can gain.

There have been many variants of Bellovin and Merritt's EKE protocol, as well as many alternative protocols. Recently these have included some with proven security properties. The original EKE protocol does not specify the encryption algorithm to be used or how to transform the password to a relevant key. Some later protocols have used specific algorithms which typically require a means to map the shared password to an element of the group in which a Diffie–Hellman exchange takes place. Most protocols have variants in which the server stores only an image of the client password rather than the password itself. This means that compromise of the server does not automatically compromise the password; however, it will allow a brute force searching attack so this is not necessarily a significant advantage. Some authors have explored further the situation in which the client has access to a public key of the server and there are now protocols with provable security in this situation too.

[1] Halevi and Krawczyk [140] provide formal arguments in support of this view.

The majority of the proposed protocols use Diffie–Hellman-based key agreement with the shared password used for authentication purposes. We therefore require notation similar to that used in Chap. 5. The notation used in this chapter is shown in Table 7.1. Many of the protocols examined in this chapter are client–server protocols for which the actions of the client are different from those of the server. Specifically, the client always possesses the plain password π while the server sometimes stores only an image of π under some one-way function. In order to avoid confusion we always assume that principal A is the client and principal B is the server.

Table 7.1. Notation for password-based protocols

π	A key of short length, such as a password, owned by A.
p	A large prime (usually at least 1024 bits).
q	A prime with $q \mid p-1$.
G	A subgroup of $\mathbb{Z}_p^*$. G is often a subgroup of order q, but sometimes is equal to $\mathbb{Z}_p^*$.
g	A generator of G.
r_A, r_B	Random integers, typically of the same size as the order of G, chosen by A and B respectively.
t_A, t_B	g^{r_A} and g^{r_B} respectively. All computations take place in $\mathbb{Z}_p$.
x_A, x_B	The private long-term keys of A and B respectively.
y_A, y_B	The public keys of A and B, g^{x_A} and g^{x_B} respectively. These public keys will have to be certified in some standard way which we do not consider here.
Z_{AB}	The shared secret calculated by the principals.
K_{AB}	The derived session key.
S_{AB}	The static Diffie–Hellman key of A and B.
$H_i(.)$	One-way hash functions. Functions $H_1, H_2, \ldots$ are often assumed to be independent random functions. Certain protocols may require specific properties and may specify particular functions.

In the next section we examine in some detail the version of EKE based on Diffie–Hellman key exchange and later protocols derived from it. Section 7.3 is concerned with EKE variants in which the server stores only an image of the password; these are often called *augmented* password-based protocols. Then in Sect. 7.4 we look at protocols in which a server shares passwords with two users and establishes a key between them. Most protocols in this chapter are based on Diffie–Hellman, but in Sect. 7.5 some protocols based on RSA are described. Section 7.6 is concerned with protocols that make use of a server public key and we also look at some protocols that do not fit into

any of the above sections. In the final section we make a comparison of the properties achieved and resources used in a selection of the protocols.

7.2 Encrypted Key Exchange Using Diffie–Hellman

In this section we examine the Encrypted Key Exchange (EKE) protocol of Bellovin and Merritt [38] as applied to Diffie–Hellman key exchange. Although the original protocol has potential weaknesses and lacks a proof of security, it is instructive to understand what may go wrong with such protocols: completely different attacks are possible from those applicable to protocols with strong keys. We later examine the many variants and extensions that have subsequently been developed from the original idea. Steiner *et al.* [306] give a specification of how to integrate EKE into the TLS protocol including details of how to use the symmetric encryption algorithm.

7.2.1 Bellovin–Merritt's Original EKE

The general idea of EKE is to transmit ephemeral public keys encrypted using the password as a shared key. Only parties that know the password should be able to complete the protocol. This idea can be applied to ephemeral keys from different public key schemes. Here we consider only Diffie–Hellman key exchange in which both principals choose an ephemeral public key; the password is used to encrypt the ephemeral keys as shown in Protocol 7.1.

Shared information: Shared password π. Security parameter L.

A		B
$r_A \in_R \mathbb{Z}_p$		
$t_A = g^{r_A}$	$\xrightarrow{A, \{t_A\}_\pi}$	$r_B \in_R \mathbb{Z}_p$
		$Z_{AB} = t_A^{r_B}$
		$t_B = g^{r_B}$
		$n_B \in_R \{1, \ldots, 2^L\}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{\{t_B\}_\pi, \{n_B\}_{K_{AB}}}$	
$n_A \in_R \{1, \ldots, 2^L\}$		
	$\xrightarrow{\{n_A, n_B\}_{K_{AB}}}$	Verify n_B
Verify n_A	$\xleftarrow{\{n_A\}_{K_{AB}}}$	

Protocol 7.1: Diffie–Hellman-based EKE protocol

As in basic Diffie–Hellman key agreement, the shared secret is $Z_{AB} = g^{r_A r_B}$ although the key derivation function to obtain the session key K_{AB} from Z_{AB} is not specified. Protocol 7.1 requires two exponentiations by each party, which is the same as ordinary Diffie–Hellman. We will see below that there are versions with fewer message passes.

Symmetric Encryption Algorithm

The choice of symmetric encryption algorithm using π was left flexible by Bellovin and Merritt. They suggested that many choices are acceptable, even ones that are 'quite weak'. However, it now seems clear that use of encryption functions without special properties prevents security proofs being obtained and allows attacks in certain cases.

Bellovin and Merritt introduced the notion of *partition attacks* against EKE. The idea is that an adversary guessing the password can attempt to decrypt $\{t_A\}_\pi$ and $\{t_B\}_\pi$ and examine whether the resulting plaintext is a valid Diffie–Hellman ephemeral value. If not then the guessed password is incorrect and can be discarded. Given several runs of the protocol, successive 'partitions' of the passwords into valid and invalid sets may be obtained. The success of partition attacks can depend on two factors: the symmetric encryption used, and the parameters defining the group G in which the protocol works.

Consider an example in which the symmetric encryption algorithm is a conventional block cipher (the key can be derived from the password using a suitable hash function). If $G = \mathbb{Z}_p^*$ then decryption of $\{t_A\}_\pi$ using a candidate password can be assumed to be a random string of the appropriate block length. This string must be interpreted as an element of $\mathbb{Z}_p^*$. During encryption any padding that must be included due to the algorithm block length can be chosen randomly (as suggested by Bellovin and Merritt). Even so, there will be some bit-strings that cannot occur as plaintexts and these allow some passwords to be discarded. This is because if decryption with a candidate password results in a string whose value is greater than p then that candidate is invalid.

In order to make partition attacks harder, Bellovin and Merritt suggest to choose p to be slightly less than a power of 2. In this way very few candidate passwords will be invalid. Jaspan [170] conducted an analysis which concludes that it is adequate in practice to ensure that $1 - (p/2^n) < 10^{-4}$, where 2^n is the smallest power of 2 greater than p. However, it seems preferable to choose the symmetric encryption algorithm to be matched to the group so as to completely eliminate such attacks; we will explain below how this may be done.

Omitting Symmetric Encryption

Bellovin and Merritt suggested that either of the two encryptions using π in Protocol 7.1 could be omitted. Subsequent authors have shown that this can

result in weaknesses depending on the precise format of the messages used for authentication.

First suppose that the encryption using π is omitted from the first message so that this message becomes: A, t_A. This change does not appear to help a passive adversary and still prevents K_{AB} from being found without knowledge of π. However, Patel [262] shows how an active adversary masquerading as A may be able to exploit this change. The adversary can send the first message by choosing r_C and setting $t_A = g^{r_C}$. Now if the authenticator $\{n_B\}_{K_{AB}}$ returned in the second message contains redundancy then the adversary can use it to eliminate any potential password $\tilde{\pi}$ by decrypting $\{t_B\}_{\pi}$ with $\tilde{\pi}$ to obtain $\tilde{t}_B$, finding the corresponding value of the session key $\tilde{K}_{AB}$ from $\tilde{Z}_{AB} = (\tilde{t}_B)^{r_C}$, and checking for the redundancy after decrypting $\{n_B\}_{K_{AB}}$ with $\tilde{K}_{AB}$. Jablon [164] noted that small subgroup attacks (see Sect. 5.2.1) may also become possible in this situation; the adversary can move t_A into any small subgroup and use the authenticator in the third message to identify the correct K_{AB} in a brute force attack. Standard precautions against small subgroup attacks can prevent this.

If encryption using π is omitted from the second message then the adversary can attempt to masquerade as B. Steiner *et al.* [307] showed that this allows an attack if the authenticators *do not* contain redundancy. The adversary chooses a random X and r_C, sets $t_B = g^{r_C}$, and sends t_B, X as the second message, which will be accepted by A. Now the adversary can decrypt $\{t_A\}_{\pi}$ with a candidate password $\tilde{\pi}$ to obtain $\tilde{t}_A$ and the corresponding session key $\tilde{K}_{AB} = (\tilde{t}_A)^{r_C}$. The adversary can then decrypt message 3 using $\tilde{K}_{AB}$ and check whether the second field equals the decryption of X using $\tilde{K}_{AB}$, in order to confirm whether $\tilde{\pi}$ is the correct password. Patel [262] also noted an attack here when there is limited redundancy in the authenticator, but this time it is necessary for the adversary to wait to see if a random authenticator is accepted by A: the adversary sends t_B, X as the second message for a random value X. If this message is accepted then the adversary knows that the redundancy is present, but if it is not then all passwords that result in the desired redundancy can be eliminated.

It may seem reasonable to conclude that it is safest not to omit either of the symmetric encryptions using π. However, a better solution is to be more careful in the use of authenticators. We now examine variants in which this is done; not only is the result a protocol with fewer rounds, but also a proof of security becomes feasible.

7.2.2 The PAK Protocol

Bellare *et al.* [29] have provided proofs that the exchange of the password-encrypted Diffie–Hellman messages at the core of EKE is a secure protocol using the Bellare–Rogaway model discussed in Chap. 2. Although they specify the key derivation function to find the session key the symmetric encryption algorithm used in the protocol is not concretely defined; it is assumed to be

an 'ideal cipher' that can be regarded as a random function (analogous to the functions used in the random oracle model). Furthermore, the decryption function must take strings of a fixed size and map them to elements of G. This may leave the implementer in some difficulty in choosing an appropriate concrete algorithm. Later MacKenzie [218] proposed concrete ways to achieve a suitable algorithm along with complete proofs.

In parallel work, Boyko *et al.* [65] specified a slightly different version of Diffie–Hellman-based EKE called PAK (Password Authenticated Key exchange). They provided a proof of security in Shoup's simulation model, and later MacKenzie provided proofs in the Bellare–Rogaway model too. The encryption function used is simply multiplication in the Diffie–Hellman group.

PAK uses a prime p of the form $p = rq + 1$, with r and q relatively prime. The Diffie–Hellman key exchange takes place in the subgroup G of order q. The original PAK protocol uses four hash functions. It is important that these functions are different and the formal proof assumes they are independent random functions. In practice we may define the different hash functions by prepending a different fixed string to the input and then using a standard hash function. For example, we may define $H_i(x) = H(\mathrm{ASCII}(i), x)$, where H is SHA-1 [245] and ASCII(i) is the ASCII representation of i. Protocol 7.2 shows the basic version of PAK. Output from H_1 is treated as an element of $\mathbb{Z}_p^*$ and is used to map π to an element of G which is an important part of defining the symmetric encryption algorithm.

A key feature of Protocol 7.2 is that the element P in the group G is derived from the password π using the formula $P = H_1(A, B, \pi)^r$. In typical applications the computing device of A will not have the value π available beforehand and so this calculation is part of the computational requirements for A. (In contrast we may assume that B is a server which can store the derived value of P.) Since typical lengths of q and r may be 160 bits and 864 bits respectively, this calculation is more expensive than the subsequent Diffie–Hellman exchange. The computational requirements may be optimised for different applications by choosing a larger size for q so that the size of r becomes smaller.

The PAK protocol was proven by Boyko *et al.* [65] to be secure in Shoup's simulation model assuming the hash functions act as random oracles. Originally this result relied on the Decision Diffie–Hellman assumption but later MacKenzie [218] showed that the ordinary (computational) Diffie–Hellman assumption is sufficient. We may regard the PAK protocol as a variant of the original Diffie–Hellman-based EKE protocol with the following instantiations and modifications:

- the key derivation function is specified;
- symmetric encryption is defined as multiplication in G by the image P of π;
- symmetric encryption of the ephemeral Diffie–Hellman key in the second message is omitted;

Shared information: Generator g of G where $p - 1 = qr$. Hash functions H_1, H_2, H_3, H_4.
Information held by A: Password π.
Information held by B: Derived password image $P = H_1(A, B, \pi)^r$.

A		B
$P = H_1(A, B, \pi)^r$		
$r_A \in_R \mathbf{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A P$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$t_A = m/P$
		$r_B \in_R \mathbf{Z}_q$
		$t_B = g^{r_B}$
		$Z_{AB} = t_A^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(A, B, m, t_B, Z_{AB}, P)$
$k \stackrel{?}{=} H_2(A, B, m, t_B, Z_{AB}, P)$		
$k' = H_3(A, B, m, t_B, Z_{AB}, P)$	$\xrightarrow{k'}$	$k' \stackrel{?}{=} H_3(A, B, m, t_B, Z_{AB}, P)$
$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$		$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$

Protocol 7.2: PAK protocol

- a simplified authentication mechanism is used which allows a more efficient protocol.

It is interesting to consider how these changes ensure that the potential weaknesses in Diffie–Hellman EKE are avoided. The choice of symmetric encryption is very simple and matched to the group G. (Although a modular multiplication may be computationally more expensive than encryption with a block cipher, in practice the additional effort over the required exponentiations is very small.) In the partition attack described above the adversary attempts decryption with candidate passwords π'. With this natural choice of encryption, every π' maps to an image $P' \in G$ and attempted decryption of $t_A P$ results in $t_A P/P'$ which will always lie in G. Therefore the partition attack cannot even discount a single candidate π'. If the adversary tries to exploit the omission of the symmetric encryption with π in the second message by sending t_B, X for a random X then with overwhelming probability A will reject the message. Instead the adversary can calculate a correct value k for any candidate password and this would allow checking of that one password.

Boyko *et al.* [65] also specify a variant protocol of PAK without explicit authentication. The result is shown as Protocol 7.3 and is known as PPK (Password Protected Key exchange). Apart from omitting the authenticators

the difference is that both messages, instead of just the first, must now use the symmetric encryption with π. A consequence of using two encryptions is that A requires an extra exponentiation, in comparison with Protocol 7.2, to derive the extra password image P_2.

Shared information: Generator g of G where $p-1=qr$. Hash functions H_1, H_2, H_3.
Information held by A: Password π.
Information held by B: $P_1 = H_1(A,B,\pi)^r$ and $P_2 = H_2(A,B,\pi)^r$.

A		B
$P_1 = H_1(A,B,\pi)^r$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A P_1$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$t_A = m/P_1$
		$r_B \in_R \mathbb{Z}_q$
$P_2 = H_2(A,B,\pi)^r$		$t_B = g^{r_B}$
$m' \bmod p \stackrel{?}{\neq} 0$	$\xleftarrow{m'}$	$m' = t_B P_2$
$t_B = m'/P_2$		
$Z_{AB} = t_B^{r_A}$		$Z_{AB} = t_A^{r_B}$

$$K_{AB} = H_3(A,B,m,m',Z_{AB},P_1)$$

Protocol 7.3: PPK protocol

If the first message alone was protected by the password, as in Protocol 7.2, then an adversary could masquerade as B and use knowledge of the derived session key to mount a brute force searching attack. For example, the adversary could choose random $r_C \in \mathbb{Z}_q$ and send g^{r_C} to A as the second message. Then the shared secret accepted by A is $Z_{AB} = g^{r_C r_A}$. The adversary can check any potential password $\tilde{\pi}$ by calculating $\tilde{P}_1 = H_1(A,B,\tilde{\pi})^r$, $\tilde{t}_A = m/\tilde{P}_1$ and the corresponding secret $\tilde{Z}_{AB} = (\tilde{t}_A)^{r_C}$. If $\tilde{\pi}$ is the correct password then $\tilde{Z}_{AB} = Z_{AB}$, which can be checked against subsequent messages from A protected with K_{AB}.

MacKenzie [216] later designed several other variants of the PAK protocol and proved their security. These included versions of the protocol for elliptic curve groups and the XTR group [204]. Protocol 7.4 is an ingenious variant which transfers computational load from the client A to the server B. This is very useful in applications in which the client is a device, such as a smart card, with reduced computational power. The idea is that instead of moving the password image into the group G, the password is simply regarded as a

value in $\mathbb{Z}_p^*$ and is masked by a random value of order r. The result is that A no longer needs to compute the exponentiation $H_1(A, B, \pi)^r$ with the large exponent r. However, in order to remove the element of order r, B now has to compute a costly exponentiation instead.

Shared information: Generator g of G where $p - 1 = qr$. Hash functions H_1, H_2, H_3, H_4.
Information held by A: Password π.
Information held by B: Derived password image $P = H_1(A, B, \pi)$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$h \in_R \mathbb{Z}_p^*$		
$m = t_A h^q \cdot H_1(A, B, \pi)$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$t_A = ((m/P)^r)^{r^{-1} \bmod q}$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$Z_{AB} = t_A^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(A, B, m, t_B, Z_{AB}, P)$
$k \stackrel{?}{=} H_2(A, B, m, t_B, Z_{AB}, P)$		
$k' = H_3(A, B, m, t_B, Z_{AB}, P)$	$\xrightarrow{k'}$	$k' \stackrel{?}{=} H_3(A, B, m, t_B, Z_{AB}, P)$
$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$		$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, P)$

Protocol 7.4: PAK-R protocol

7.2.3 SPEKE

The SPEKE (Secure Password Exponential Key Exchange) protocol was designed by Jablon [164]. Although SPEKE, like EKE, is based on Diffie–Hellman, it has a distinctive feature that the password is used to define the base to be used for the Diffie–Hellman exchange. Jablon presents both basic and 'fully constrained' versions, the latter designed to prevent a variety of different attacks.

Protocol 7.5 describes the fully constrained SPEKE protocol, but we have used the verifiers, V_A and V_B, as defined in the version analysed by MacKenzie [217]. Jablon's original verifiers included only Z_{AB} in the hash, which allows a potential attack in which the adversary masquerades as A and makes a guess $\tilde{\pi}$ for the password π. One verifier then allows the adversary to check if $\tilde{\pi}$

is any value in the set $\{\pi, \pi^2, \pi^3, \ldots\}$. The seriousness of such an attack can be debated since there may be few potential passwords in this sequence, but MacKenzie's verifiers rule it out altogether.

Shared information: Subgroup G of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Shared derived password image $P = \pi^2$ where π is interpreted as an element of $\mathbb{Z}_p^*$. Security parameter L.

A		B
$r_A \in_R \{1, \ldots, 2^L\}$		
$t_A = P^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \{1, \ldots, 2^L\}$
		$Z_{AB} = t_A^{r_B}$
		Check $Z_{AB} \notin \{-1, 0, 1\}$
		$t_B = P^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, V_B}$	$V_B = H_1(t_A, t_B, Z_{AB}, \pi)$
Check $Z_{AB} \notin \{-1, 0, 1\}$		
Verify V_B		
$V_A = H_2(t_A, t_B, Z_{AB}, \pi)$	$\xrightarrow{V_A}$	Verify V_A
$K_{AB} = H_3(Z_{AB})$		$K_{AB} = H_3(Z_{AB})$

Protocol 7.5: SPEKE protocol

The prime p and subgroup G are chosen so that $q = (p-1)/2$ is prime and G is of order q. The password π is regarded as a number in $\mathbb{Z}_p^*$ and then $P = \pi^2$ is guaranteed to lie in G and have order q (assuming that π is not equal to 1, -1, or 0). The value P is used as the generator of G for protocol runs involving π. Apart from this special way of defining the group generator, the protocol is exactly the basic Diffie–Hellman key exchange with key confirmation. The shared secret is therefore $Z_{AB} = P^{r_A r_B}$ and forward secrecy is provided. The check that Z_{AB} is not in the set $\{-1, 0, 1\}$ ensures that small subgroup attacks are avoided. In comparison with Protocol 7.2, calculation of P from π is much simpler so that we can almost ignore it in assessing A's computational requirements. The cost of the Diffie–Hellman exchange calculations depends on the size of the exponents r_A and r_B. Typical lengths might be 160 bits.

The original version of SPEKE does not have a proof of security, but later MacKenzie [217] did provide a proof of a slight variant using Shoup's simulation model. This proof relies on the difficulty of a non-standard decision problem (called the 'Inverted-Additive Diffie–Hellman problem'). It shows that at most two passwords can be tested per login attempt, whereas it is proven that only one password test is possible for the PAK protocol. The differences between the protocol used by MacKenzie and Protocol 7.5 are:

- P is defined to include the identities of A and B: $P = (H(A, B, \pi))^2$;
- the hashes used to form the session key include the identities of A and B, the exchanged messages t_A and t_B, the password π, as well as Z_{AB};
- the exponents are chosen randomly in $\mathbb{Z}_q$.

An important practical consequence of the last difference is that exponents are the same size as q. Since $q = (p-1)/2$ this would typically make the exponents around 1024 bits, a significant overhead compared with exponents of size 160 bits suggested by Jablon. Versions of SPEKE suitable for use on elliptic curves, which are potentially much more efficient, are specified in the emerging IEEE P1363.2 standard [155].

A flawed variant of SPEKE appeared in the original paper of Jablon [164]. In this variant the derived password is $P = g^{\pi}$ which, as in Protocol 7.5, is used as generator of the group G in a Diffie–Hellman exchange. The shared secret is again $Z_{AB} = P^{r_A r_B}$. A password guessing attack was found by Jablon and others and resulted in the deletion of this option from later papers on SPEKE [165]. An essentially identical protocol also appears in the paper of Bakhtiari *et al.* [23]. A protocol designed by Seo and Sweeney [293] known as SAKA (Simple Authenticated Key Agreement) was an unfortunate re-discovery of almost the same protocol. The main difference is that the shared secret is $Z_{AB} = g^{r_A r_B}$ but this seems to have little influence. Mitchell [236] showed that a password guessing attack is still possible. A number of other authors [319, 194, 209] have also found various attacks concerned with the explicit authentication exchange phase of SAKA.

A protocol by Kaufman and Perlman [179] is connected with SPEKE only in that the password is used to generate the parameters for the Diffie–Hellman exchange. However, the difference is that in this case the password is used as a seed to generate the prime modulus p, and this is done in such a way that 2 is a generator (at least with high probability) of $\mathbb{Z}_p^*$. The protocol is known as PDM (for Password Derived Moduli). Like SPEKE, it is essentially basic Diffie–Hellman with key confirmation. A computational overhead is the time taken to reconstruct p at both ends, which entails testing for primality (indeed testing for a strong prime) starting from the seed value. In addition, no formal security proof is provided.

7.2.4 Katz–Ostrovsky–Yung Protocol

The protocol of Katz *et al.* [178] does not strictly belong in this section, since it is not really an EKE variant. However, the algebraic setting is the same as other protocols in this section, and there are other similarities too. The protocol has a strong theoretical significance because of its security proof. The proof of security for all the PAK protocol variants uses the random oracle model. From a theoretical viewpoint, eliminating such an assumption is very desirable. The protocol of Katz *et al.* is proven secure in the Bellare–Rogaway model without using random oracles, assuming the Decision Diffie–Hellman

problem is hard. The encryption algorithm used in the protocol is a variant of the Cramer–Shoup algorithm [97], chosen because it is proven to provide non-malleability without itself using random oracles. As shown in Protocol 7.6, there are a number of public parameters.

Shared information: Generators g_1, g_2, h, c, d of G where $p - 1 = qr$. Hash function H. Password π.

A		B
Generate ephemeral signature keys PK, SK		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g_1^{r_A}$		
$u_A = g_2^{r_A}$		
$v_A = h^{r_A} g_1^{\pi}$		
$\alpha = H(A, PK, t_A, u_A, v_A)$		
$w_A = (cd^{\alpha})^{r_A}$		
	$\xrightarrow{A, PK, t_A, u_A, v_A, w_A}$	
		$x_B, y_B, z_B, z'_B, r_B \in_R \mathbb{Z}_q$
		$\alpha = H(A, PK, t_A, u_A, v_A)$
		$E = g_1^{x_B} g_2^{y_B} h^{z_B} (cd^{\alpha})^{z'_B}$
		$t_B = g_1^{r_B}$
		$u_B = g_2^{r_B}$
		$v_B = h^{r_B} g_1^{\pi}$
		$\beta = H(B, E, t_B, u_B, v_B)$
		$w_B = (cd^{\beta})^{r_B}$
	$\xleftarrow{B, E, t_B, u_B, v_B, w_B}$	
$\beta = H(B, E, t_B, u_B, v_B)$		
$x_A, y_A, z_A, z'_A \in_R \mathbb{Z}_q$		
$K = g_1^{x_A} g_2^{y_A} h^{z_A} (cd^{\beta})^{z'_A}$		
	$K, \xrightarrow{Sig_{SK}(\beta, K)}$	
		Verify signature
$Z_{AB} = E^{r_A} t_B^{x_A} u_B^{y_A} (v_B / g_1^{\pi})^{z_A} w_B^{z'_A}$		$Z_{AB} = K^{r_B} t_A^{z_B} u_A^{y_B} (v_A / g_1^{\pi})^{z_B} w_A^{z'_B}$

Protocol 7.6: Katz–Ostrovsky–Yung protocol

The first message sent from A to B consists of the encryption of the plaintext message g_1^{π} in the Cramer–Shoup variant. However, an amusing aspect of the protocol is that the encryption is turned around; the public parame-

ters g_1, g_2, h, c, d form a public key for which no corresponding private key is known. This ensures that information on the password is not leaked. But since B already knows the value g_1^{π}, he can obtain the correct parameters to form the session key. The second message returned from B is also an encryption of g_1^{π} using a different Cramer–Shoup variant but with the same public parameters. Notice that the signature sent in the third message uses a public key generated randomly by A during the protocol, and is not intended to provide entity authentication. Indeed, neither party obtains key confirmation or entity authentication. Katz *et al.* note that the usual additional checks are required, in particular that received values lie in the group G.

Although the protocol of Katz *et al.* is certainly practical, its computational requirements are significantly greater than any of the PAK variants, around 5–10 times as great depending on which variant is compared and how the computation is measured.

7.3 Augmented EKE

For many years it has been standard practice for passwords to be stored on servers under the image of a one-way function H, so that $H(\pi)$ is stored rather than π itself. In this way compromise of the password file will not compromise the passwords directly, and when a claimed password π' is submitted to the server it can be checked whether $H(\pi') = H(\pi)$. However, compromise of the password file will nevertheless allow an off-line dictionary attack to be mounted since the adversary can calculate $H(\pi')$ for any password guess π' and compare with the stored value. Therefore the benefits of this approach will depend on the application. Many designers have provided password-based key establishment protocols that require the server to store only an image of each password.

A number of protocols explicitly include the use of *salt* in the calculation of the password image. Salt is another commonly used mechanism for protecting passwords on a server. For each password π_i, a random salt value s_i is chosen and the pair $(s_i, H(\pi_i, s_i))$ can be stored on the server. The purpose of salt is to frustrate bulk guessing attacks if the password file stored on the server becomes compromised. Even though the salt becomes known to the adversary, any password guess can only be verified against one password image at a time.

The original EKE protocol requires that the server stores the plaintext password in order to be able to decrypt the protocol exchanges. Subsequently Bellovin and Merritt [39] designed an 'augmented' version, shown in Protocol 7.7, which requires the server to store only an image of the password.

The first four messages of this protocol are the same as Protocol 7.1 with the plain password π replaced by the hashed password $H_1(\pi)$. These messages are insufficient on their own since the user would only require $H_1(\pi)$ in order to complete the protocol so there is no effective difference from the original EKE. Therefore the fifth message is added which consists of a signature of the

Information held by A: Password π.
Information held by B: Two images of password, $H_1(\pi)$ and $H_2(\pi)$. $H_2(\pi)$ is suitable for signature verification.

A		B
$r_A \in_R \mathbb{Z}_p$		
$t_A = g^{r_A}$	$\xrightarrow{A, \{t_A\}_{H_1(\pi)}}$	$r_B \in_R \mathbb{Z}_p$
		$Z_{AB} = t_A^{r_B}$
		$t_B = g^{r_B}$
		$n_B \in_R \{1, \ldots, 2^L\}$
$Z_{AB} = t_B^{r_B}$	$\xleftarrow{\{t_B\}_{H_1(\pi)}, \{n_B\}_{K_{AB}}}$	
$n_A \in_R \{1, \ldots, 2^L\}$		
	$\xrightarrow{\{n_A, n_B\}_{K_{AB}}}$	Verify n_B
Verify n_A	$\xleftarrow{\{n_A\}_{K_{AB}}}$	
	$\xrightarrow{\{Sig_\pi(K_{AB})\}_{K_{AB}}}$	Verify signature using $H_2(\pi)$

Protocol 7.7: Augmented Diffie–Hellman-based EKE protocol

shared secret constructed using π as the secret key and so that B can hold the corresponding public key. Bellovin and Merritt did not make an explicit choice for the signature, but an example could be an ElGamal signature [106] with private key π and public verification key $H_2(\pi) = g^{\pi}$. (It is possible to allow $H_1 = H_2$ but if, as in this example, H_2 requires an exponentiation, computation is reduced for A if H_1 is a simple hash function.)

Steiner *et al.* [307] have pointed out that an adversary who is able to obtain an old K_{AB} can decrypt the final message flow of Protocol 7.7 and attempt to verify the signature using a guessed π' value and therefore mount a dictionary attack. This protocol is therefore arguably weaker than the original in the usual scenario where old session keys may be compromised. However, this problem could be avoided by using a different key derived from Z_{AB} to protect the protocol messages, instead of using the session key K_{AB} for this purpose.

A variant of PAK, called PAK-X, was designed by Boyko *et al.* [65] and can also be considered a variant of augmented EKE. The protocol is very similar to Protocol 7.2 but incorporates an additional verifier. Later MacKenzie [216] devised a conceptually simpler version called PAK-Y shown in Protocol 7.8. The first two messages are essentially identical to those of Protocol 7.2, but in the last message A sends the Schnorr signature [291] of the fields

A, B, m, t_B, Z_{AB}, V using the derived password v as the private key. This allows B to verify that A possesses v using the corresponding public key V.

Information held by A: Password π from which is obtained $v = H_0(A, B, \pi)$. (In order to ensure a unique verifier per pair, identities may be ordered and then $v = H_0(\min(A, B), \max(A, B), \pi)$.) Hash functions H_1, H_2, H_3, H_4.
Information held by B: Derived password image $V = g^v$.
Shared information: Group G of prime order $q = (p-1)/r$ and generator g of G.

A		B
$V = g^v$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		$P = H_1(A, B, V)^r$
$m = t_A \cdot H_1(A, B, V)^r$	$\xrightarrow{m}$	$m \bmod p \stackrel{?}{\neq} 0$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$Z_{AB} = (m/P)^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(A, B, m, t_B, Z_{AB}, V)$
$k \stackrel{?}{=} H_2(A, B, m, t_B, Z_{AB}, V)$		
$c \in_R \mathbb{Z}_q$		
$a = g^c$		
$e = H_3(A, B, m, t_B, Z_{AB}, a, V)$		
$s = c - ev \bmod q$	$\xrightarrow{e, s}$	$e \stackrel{?}{=} H_3(A, B, m, t_B, Z_{AB}, g^s V^e, V)$
$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, V)$		$K_{AB} = H_4(A, B, m, t_B, Z_{AB}, V)$

Protocol 7.8: PAK-Y protocol

The augmented versions of PAK can also be combined with the 'R' version shown in Protocol 7.4 in order to shift computational requirements from A to B. The only change required in Protocol 7.8 is to replace the first message with that of Protocol 7.4 and adjust the corresponding computation by B.

Recently MacKenzie [218] proposed a generalisation of PAK-Y, called PAK-Z, which allows any digital signature to take the place of the Schnorr signature used in PAK-Y. The server B stores the public and private keys for the signature algorithm and sends the private signing key to A masked with the shared secret. Then A returns a signature on t_B as the third message. As well as allowing any signature to be used, PAK-Z uses exactly the same verifier k as used in Protocol 7.2 so that the result is a more modular approach to the set of PAK protocols.

7.3.1 B-SPEKE

Jablon [165] discussed the method used by Bellovin and Merritt to augment EKE and proposed an alternative. He pointed out that Bellovin and Merritt's technique could be applied generally and in particular to his own SPEKE protocol leading to a variant he called A-SPEKE. His own augmentation technique he called the 'B' extension.

Instead of employing a signature to prove knowledge of the password, as used in Bellovin and Merritt's 'A' extension, the 'B' variant uses an additional Diffie–Hellman exchange. The server B stores two password images as before: one is the square of the hash, $P = H(\pi)^2$, and the other is $V = g^{\pi}$. During the protocol B sends an additional challenge $g^{\tilde{r}_B}$ to A and A must respond by showing knowledge of the Diffie–Hellman key $\tilde{Z}_{AB} = g^{\pi\tilde{r}_B}$. A cannot simply send $\tilde{Z}_{AB}$ to B since this value could be used together with the challenge to form a dictionary attack on π. However, if A returns a hash containing both Z_{AB} and $\tilde{Z}_{AB}$, the hash can be used both to check knowledge of π and for confirmation of Z_{AB}. Employing this technique to the SPEKE protocol leads to B-SPEKE, Protocol 7.9.

Information held by A: Password π, interpreted as an element of $\mathbf{Z}_q$.
Information held by B: Derived password image $V = g^{\pi}$. Derived password image $P = H(\pi)^2$ where $H(\pi)$ is interpreted as an element of $\mathbf{Z}_p^*$.
Shared information: Subgroup G of $\mathbf{Z}_p^*$ of prime order $q = (p-1)/2$. Hash functions H, H_0, H_1, H_2. Security parameter L.

A		B
$P = H(\pi)^2$		
$r_A \in_R \{1,\ldots,2^L\}$		
$t_A = P^{r_A}$	$\xrightarrow{t_A}$	$r_B, \tilde{r}_B \in_R \{1,\ldots,2^L\}$
		$Z_{AB} = t_A^{r_B}$
		Check $Z_{AB} \notin \{-1,0,1\}$
		$t_B = P^{r_B}, \tilde{t}_B = g^{\tilde{r}_B}$
	$\xleftarrow{t_B, V_B, \tilde{t}_B}$	$V_B = H_1(t_A, t_B, Z_{AB}, P)$
$Z_{AB} = t_B^{r_A}$		$\tilde{Z}_{AB} = V^{\tilde{r}_B}$
Check $Z_{AB} \notin \{-1,0,1\}$		
Verify V_B		
$\tilde{Z}_{AB} = (\tilde{t}_B)^{\pi}$		
$V_A = H_2(t_A, t_B, Z_{AB}, \tilde{Z}_{AB}, P)$	$\xrightarrow{V_A}$	Verify V_A
$K_{AB} = H_0(Z_{AB})$		$K_{AB} = H_0(Z_{AB})$

Protocol 7.9: B-SPEKE protocol

Jablon is not explicit about the construction of the verifiers V_A and V_B in Protocol 7.9. We have adapted the verifier format used in Protocol 7.5. In any case, the proof of MacKenzie for Protocol 7.5 does not cover Protocol 7.9. Jablon [165] provides a number of variants of Protocol 7.9 to optimise various features; for example, he presents a version designed to minimise the total time by allowing various calculations to take place in parallel.

7.3.2 SRP Protocol

The SRP (Secure Remote Password) protocol was designed by Wu [332] as a more efficient alternative to the original augmented EKE. The server B holds a salt value s and the password image $V = g^{H(s,\pi)}$. Protocol 7.10 shows an optimised version of SRP, minimising the number of messages.

Shared information: Subgroup G of $\mathbf{Z}_p^*$ of prime order $q = (p-1)/2$. Hash function H.
Information held by A: Password π.
Information held by B: Password image $V = g^{H(s,\pi)}$. Salt value s.

A		B
$r_A \in_R \mathbf{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{A, t_A}$	$u, r_B \in_R \mathbf{Z}_q$
		$Z_{AB} = (t_A V^u)^{r_B}$
		$K_{AB} = H(Z_{AB})$
$V = g^{H(s,\pi)}$	$\xleftarrow{S, s, u}$	$S = V + g^{r_B}$
$Z_{AB} = (S - V)^{r_A + uH(s,\pi)}$		
$K_{AB} = H(Z_{AB})$		
$M_1 = H(A, B, K_{AB})$	$\xrightarrow{M_1}$	Verify M_1
		$M_2 = H(A, M_1, K_{AB})$
Verify M_2	$\xleftarrow{M_2}$	

Protocol 7.10: SRP protocol

The shared secret is $Z_{AB} = g^{r_A r_B} \cdot V^{u r_B}$ which is unusual in that it is asymmetric with regard to the inputs of A and B. Another unusual feature is the public random value u chosen by B and included in the derivation of Z_{AB}. The purpose of u is to ensure that A knows the value of π and not just its image V. If A could know u in advance then she could send $g^{r_A} V^{-u}$ in the first message instead of t_A and then B would calculate $Z_{AB} = g^{r_A r_B}$. This means that A could complete the protocol only with knowledge of V. In order

to avoid this possibility it is evident that u cannot be a fixed value. However, for efficiency reasons, Wu suggests that u may be defined as a public function of S which would allow it to be omitted from the exchanged messages.

The symmetric encryption algorithm used with the password is addition modulo p. Forward secrecy is provided since the ephemeral Diffie–Hellman key, $g^{r_A r_B}$, is needed in order to find Z_{AB}. The main drawback of SRP is a lack of a formal security proof. However, Wu did point out that security against passive eavesdroppers is provided in the sense that an oracle that can find the session key from the exchanged messages t_A, S and u, is able to solve the Diffie–Hellman problem.

A minor weakness in Protocol 7.10, whose discovery is attributed by MacKenzie [217] to Bleichenbacher, allows the adversary to eliminate two passwords, π_1 and π_2, with each protocol run. An adversary masquerading as B can exploit the symmetry of the value S by choosing $V_1 = g^{H(s,\pi_1)}$ and $V_2 = g^{H(s,\pi_2)}$ and sending $S = V_1 + V_2$ instead of a correctly formed S in the second message. Then if π_1 is the correct password the adversary can check that $Z_{AB} = V_2^{r_A + uH(s,\pi_1)}$ while a symmetric test can be used for π_2.

The calculation of V must be explicitly made by A during the protocol, but Wu implies that $H(s,\pi)$ can be a 'tiny' exponent since the entropy of π is small. Wu regards as optional whether the values p and g are fixed or are sent as part of the protocol itself (the former situation is intended in Protocol 7.10). If p and g are sent as parameters by B at the start of the protocol it is essential that A checks the primality of $q = (p-1)/2$ and that g has order q.

Kwon and Song [198] have proposed a similar protocol to SRP in which the image of the password is similarly added to one of the ephemeral public keys in order to encrypt it.

7.3.3 AMP Protocol

The AMP (Authentication via Memorable Password) protocol of Kwon [195] is a variant of Diffie–Hellman-based EKE that employs a number of different features found in other protocols. Like the PAK family, Protocol 7.11 masks one of the Diffie–Hellman exchanges by multiplying it with a derived version of the password. However, there is a different method to calculate the shared secret which depends in an unusual way on the exchanged values. The shared secret is $Z_{AB} = g^{r_B(r_A+e)}$ where e is a hash of the exchanged values t_A and T together with the principal identities.

Protocol 7.11 lacks a formal proof of security. It turns out that, like Protocol 7.10, an adversary can eliminate two passwords in one protocol run[2]. To do this the adversary chooses $t_A = g^{-H_1(s,\pi_1)}$ in the first message where π_1 is the first password guess. If this guess is correct then the value T returned

[2] This attack was described by Michael Scott in a message on the IEEE P1363 mailing list in July 2001.

Shared information: Subgroup G of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2r$. Four hash functions H, H_1, H_2 and H_3.
Information held by A: Password π.
Information held by B: Password image $V = g^{H_1(s,\pi)}$. Salt value s.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{A, t_A}$	$r_B \in_R \mathbb{Z}_q$
		$T = (t_A V)^{r_B}$
	$\xleftarrow{s, T}$	
$x = H_1(s, \pi)$		
$e = H_2(t_A, T, A, B)$		$e = H_2(t_A, T, A, B)$
$Z_{AB} = T^{(r_A+e)/(r_A+x)}$		$Z_{AB} = (t_A g^e)^{r_B}$
$K_{AB} = H_3(Z_{AB})$		$K_{AB} = H_3(Z_{AB})$
	$\xrightarrow{H(t_A, K_{AB})}$	Verify hash
Verify hash	$\xleftarrow{H(T, K_{AB})}$	

Protocol 7.11: AMP protocol

by B will be equal to 1. If the guess is incorrect then the adversary can still continue the protocol and include a different guess in the calculation of Z_{AB}.

The AMP protocol requires only two exponentiations for both client and server. On the client side this is two exponentiations fewer than PAK-Y, which has a formal proof of security. Both B-SPEKE and SRP also use two main exponentiations but have additional exponentiations with 'small' exponents. In contrast AMP requires an inversion modulo q. Notice that AMP can use a value of q of 160 bits so that exponents can be efficiently reduced before exponentiation. Forward secrecy is provided since the password does not influence the value of the session key.

7.4 Three-Party EKE

The protocols we have examined so far in this chapter are appropriate for the situation when a client wishes to connect securely to a server with whom the password (or an image of the password) is shared. We now consider some protocols designed to allow two users to establish a new key through the cooperation of a mutually trusted server with whom they both share a short key or password. This class of password-based protocols has been largely neglected recently, probably due to the requirement for the on-line server. However, as well as their historical interest, it seems worthwhile to consider such protocols

due to potential efficiency gains. An alternative to using a special-purpose protocol is to run two two-party protocols between the server and each of the two principals; the two secure channels established can be then used to distribute a session key.

In this section we examine three-party EKE-type protocols; in Sect. 7.6.1 below we consider three-party protocols in which the users have knowledge of the server's public key.

7.4.1 GLNS Secret Public Key Protocols

Gong, Lomas, Needham and Saltzer (GLNS) [130] (and the same authors earlier with names permuted [210]) published the first password-based protocols for a variety of authentication and key establishment scenarios. We call their set of proposals the GLNS protocols. The majority of their protocols assume that users have knowledge of the server public key – these protocols are examined in Sect. 7.6.1 below. Here we examine their protocols which use 'secret public keys'; these are ephemeral asymmetric keys which might normally be made public but, in order to prevent guessing attacks, are kept secret from eavesdroppers on the protocol. Gong *et al.* [130] provided variants of their protocols for both the three-party and the two-party situations.

The GLNS three-party secret public key protocol is shown in Protocol 7.12. Their two-party 'Direct Authentication Protocol' uses similar ideas by essentially combining the roles of A and S. In the initial two messages the server S sends encrypted ephemeral public keys K_S for A and K'_S for B. Subsequently these are used by A and B to encrypt the request for a session key in messages 3 and 4; in these messages $E_S(.)$ denotes encryption with K_S and $E'_S(.)$ denotes encryption with K'_S.

S has passwords π_A and π_B. S chooses random value n_S and ephemeral encryption keys K_S and K'_S.
A has password π_A. A chooses random values n_A, n'_A, c_A, r_A.
B has password π_B. B chooses random values n_B, n'_B, c_B, r_B.

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: A, B, n_S, \{K_S\}_{\pi_A}, \{K'_S\}_{\pi_B}$
3. $A \rightarrow B: E_S(A, B, n_A, n'_A, c_A, \{n_S\}_{\pi_A}), n_S, r_A, \{K'_S\}_{\pi_B}$
4. $B \rightarrow S: E_S(A, B, n_A, n'_A, c_A, \{n_S\}_{\pi_A}), E'_S(B, A, n_B, n'_B, c_B, \{n_S\}_{\pi_B})$
5. $S \rightarrow B: \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{n_B, K_{AB} \oplus n'_B\}_{\pi_B}$
6. $B \rightarrow A: \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{H_1(r_A), r_B\}_{K_{AB}}$
7. $A \rightarrow B: \{H_2(r_B)\}_{K_{AB}}$

Protocol 7.12: GLNS secret public key protocol

Gong *et al.* remark on the similarity with the Otway–Rees protocol (3.22), at least in terms of the general structure. But there are a number of additional fields included in Protocol 7.12 which are worth examining. A (and B similarly) chooses two nonces n_A and n'_A which are transmitted to the server. The first of these is returned with K_{AB} with the usual purpose of allowing A to verify the freshness of this key. The second nonce is used to mask K_{AB}; its purpose is to prevent the adversary from performing a password guessing attack. If n'_A is omitted from message 5 then the adversary can decrypt the message with a candidate password $\tilde{\pi}_A$ and obtain a candidate session key $\tilde{K}_{AB}$. The adversary can then decrypt the second encrypted field in message 6 using $\tilde{K}_{AB}$ and check for the correct $H_1(r_A)$ value, in order to confirm whether $\tilde{\pi}_A$ is the correct password of A. The encrypted fields using K_{AB} in the final two messages are intended to provide key confirmation.

The value c_A (and c_B symmetrically) is a random 'confounder' used to ensure that encryption with K_S in message 4 is randomised. If c_A were not used, and the public key encryption were deterministic, then an adversary who obtained an old K_{AB} value could test a guess $\tilde{\pi}$ for π_A as follows. First the field $\{K_S\}_{\pi_A}$ in message 2 is decrypted with $\tilde{\pi}$ to obtain a possible public key $\tilde{K}_S$. Candidate values for n_A and n'_A can also be obtained by decrypting the first field in message 6 with $\tilde{\pi}$ (remember that we assume the adversary knows K_{AB}). Then all the required inputs are available to re-encrypt the first field in message 3, and check this against the actual value sent in order to verify whether $\tilde{\pi}$ is correct. The 'confounders' c_A and c_B can be omitted by requiring the asymmetric encryption algorithm to provide semantic security, since this ensures that the encryption is randomised. Furthermore, the asymmetric encryption algorithm must provide non-malleability; otherwise the protocol is easily defeated by an adversary C who can change the encrypted name of B to C in message 4 and hence masquerade as B to A.

Just as we have seen previously with Diffie–Hellman-based EKE, choice of the correct encryption algorithms is a crucial matter. Patel [262] demonstrates that use of RSA as the asymmetric encryption algorithm in Protocol 7.12 allows an active attack similar to the one against OKE in Sect. 7.5.2 below. However, Gong *et al.* did specify that the encryption algorithm must have the property that any random number could be a public key, a property not enjoyed by RSA. A protocol using an algorithm that matches the password representation to the encryption algorithm, such as used in PAK, appears more promising.

Although Protocol 7.12 seems to be regarded in the literature as a fundamentally sound one, we note that an attack is possible unless specific precautions are taken by users in an implementation. To perpetrate the attack the adversary masquerades as the server in order to obtain encrypted messages from A and/or B which can be used to verify password guesses. Specifically the adversary can guess values for π_A and π_B, generate new ephemeral keys K_S and K'_S, and use the guessed passwords to form a possibly correct message 2. This message can be sent back to A following a protocol initiation by A.

The adversary can collect the subsequent message 4 intended by B for the server. Now the adversary can detect if either of the password guesses is correct, since if so the identifiers A and B will be present in the messages when decrypted with the generated private keys. Further guesses can be verified or discarded in the same way. Notice that A must accept a random value for the field $\{K_S\}_{\pi_A}$ in message 2; otherwise an off-line guessing attack is possible.

If the above attack is to be prevented two precautions must be taken in any implementation. Firstly, failed protocol attempts must be logged by users and the password disabled after a small number of failures. Although such precautions are usually taken for failed logins at a server, it is less usual for this matter to be considered for users. Secondly, it must not be possible for the adversary to start multiple parallel runs of the protocol, since otherwise the correct passwords can be found before any protocol run has failed. It seems likely that in some applications users would be required to enter the password for each protocol run, but in others the user's computing device might cache the password and reuse it for multiple runs.

Tsudik and van Herreweghen [321], and later Gong [129], have proposed variants of Protocol 7.12 aimed at simplifying it and improving its efficiency. Protocol 7.13 shows the version of Tsudik and van Herreweghen which follows the same basic design but reduces the amount of material that needs to be encrypted with the ephemeral public keys. In distinction to Protocol 7.12 there is no attempt to provide key confirmation.

S has passwords π_A and π_B. S chooses ephemeral encryption keys K_S and K'_S.
A has password π_A. A chooses random value n_A.
B has password π_B. B chooses random value n_B.

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: \{K_S \oplus B\}_{\pi_A}, \{K'_S \oplus A\}_{\pi_B}$
3. $A \rightarrow B: A, E_S(n_A), \{K'_S \oplus A\}_{\pi_B}$
4. $B \rightarrow S: A, B, E'_S(n_B), E_S(n_A)$
5. $S \rightarrow B: n_B \oplus \{n_B \oplus K_{AB}\}_{\pi_B}, n_A \oplus \{n_A \oplus K_{AB}\}_{\pi_A}$
6. $B \rightarrow A: n_A \oplus \{n_A \oplus K_{AB}\}_{\pi_A}$

Protocol 7.13: Simplified GLNS secret public key protocol

Inclusion of the identities of A and B in the fields encrypted with π_A and π_B in message 2 is critical to the security of Protocol 7.13. This is because there is nothing else that allows the principals to know which party the session key is shared with. The nonces n_B and n_A, when XOR'd with the encrypted fields using π_B and π_A in message 5, act as 'confounders'. The outer XOR prevents B (and A similarly), having also obtained K_{AB}, from mounting a brute force guessing attack on π_A.

Ding and Horster [104] found an on-line attack on Protocol 7.13, one of several *undetectable* on-line attacks that they discovered. The important feature of such attacks is that an insider can perpetrate them in such a way that the server cannot detect that they have taken place and so cannot restrict the number of repetitions. Therefore the adversary may be able to conduct an on-line exhaustive key search. In Ding and Horster's attack on Protocol 7.13, the insider adversary B masquerades as A to initiate the protocol (the real A does not take part in the attack). The adversary collects the response from S in message 2 and makes a guess at π_A. This allows the adversary to forge a corresponding value for message 3 and hence send a trial value to S in message 4. On receipt of message 5 the adversary can verify whether the guess for π_A was correct since this means that the same K_{AB} value will be found. This attack is undetectable by S as long as there is no redundancy in the encrypted fields in message 4.

A closer look at the attack of Ding and Horster reveals that there may be no need for the adversary to be an insider in order to mount the attack. Instead the adversary can masquerade as both A and B by guessing both passwords π_A and π_B. After initiating the protocol by masquerading as A, the adversary finds candidate values for K_S and K'_S and uses these to construct message 4. On receipt of message 5 incorrect candidate passwords can be eliminated if the decrypted values of K_{AB} are different. In this version of the attack both passwords must be found and so the maximum number of trials required before success is the square of the number of passwords. Depending on the size of the password space this may still be a feasible attack.

In Gong's variant [129], the ephemeral secret public keys are generated by A and B instead of S and consequently the number of protocol messages is reduced to five. This variant is shown in Protocol 7.14, where K_A and K_B are the ephemeral public keys chosen by A and B respectively and $E_A(.)$ and $E_B(.)$ denote encryption with these keys. The confounders c_S and c'_S are chosen to prevent guessing of encrypted contents and, as elsewhere, can be omitted if a public key encryption algorithm with semantic security is used.

Ding and Horster [104] show that there is an undetectable on-line attack on Protocol 7.14 too. Again, the insider adversary B masquerades as A in a protocol run that looks normal to S. Specifically the adversary guesses π_A and chooses a value for K_A (as well as for K_B). On receipt of message 3 the adversary can decrypt both encrypted messages and see if they give the same value for K_{AB}: if so then the guess for π_A was probably correct. Notice that S must accept random values in message 2; otherwise an off-line guessing attack is possible. However, no explicit encryption algorithm for use with π_A and π_B was specified by Gong.

As with the insider attack on Protocol 7.13, this attack may be extended to an outsider attack by guessing both candidate passwords together. Indeed an outsider need guess only one of the passwords. For example, the adversary can guess π_A, choose a value for K_A, and send a message to B as if it were from A. On receipt of message 3, the adversary can detect if this guess is

S has passwords π_A and π_B. S chooses random values c_S and c'_S.
A has password π_A. A chooses random value n_A and ephemeral encryption key K_A.
B has password π_B. B chooses random value n_B and ephemeral encryption key K_B.

1. $A \rightarrow B : n_A, \{K_A\}_{\pi_A}$
2. $B \rightarrow S : n_A, \{K_A\}_{\pi_A}, n_B, \{K_B\}_{\pi_B}$
3. $S \rightarrow B : E_A(A, B, c_S, K_{AB}, \{n_A\}_{\pi_A}), E_B(B, A, c'_S, K_{AB}, \{n_B\}_{\pi_B})$
4. $B \rightarrow A : E_A(A, B, c_S, K_{AB}, \{n_A\}_{\pi_A}), \{n_A\}_{K_{AB}}, n_B$
5. $A \rightarrow B : \{n_B\}_{K_{AB}}$

Protocol 7.14: Optimal GLNS secret public key protocol

correct, since if so the identifiers A and B will be present in the encrypted field using K_A in this message, when decrypted with the generated private key. This attack is not detectable by the server S, but B will detect a failed password guess since the adversary is not able to find the correct K_{AB} value and so cannot form the correct message 5. Therefore similar remarks to those regarding the related attack on Protocol 7.12 apply.

7.4.2 Steiner, Tsudik and Waidner Three-Party EKE

Steiner *et al.* [307] proposed Protocol 7.15 as a direct generalisation of Diffie–Hellman-based EKE. They remark that, in distinction to typical three-party protocols, the server does not generate, and indeed cannot obtain, the session key. Each of A, B and S generates an ephemeral private key, r_A, r_B and r_S respectively, and the shared secret for A and B is $Z_{AB} = g^{r_A r_B r_S}$. The encrypted fields using K_{AB} in the final two messages are intended as 'authenticators' that can be used for key confirmation of Z_{AB}.

S has passwords π_A and π_B. S chooses random value r_S.
A has password π_A. A chooses random value r_A and calculates $t_A = g^{r_A}$.
B has password π_B. B chooses random value r_B and calculates $t_B = g^{r_B}$.

1. $A \rightarrow B : \{t_A \oplus B\}_{\pi_A}$
2. $B \rightarrow S : A, \{t_A \oplus B\}_{\pi_A}, \{t_B \oplus A\}_{\pi_B}$
3. $S \rightarrow B : t_A^{r_S}, t_B^{r_S}$
4. $B \rightarrow A : t_B^{r_S}, \{\text{Message 1}\}_{K_{AB}}$
5. $A \rightarrow B : \{\{\text{Message 1}\}_{K_{AB}}\}_{K_{AB}}$

Protocol 7.15: Steiner, Tsudik and Waidner three-party EKE

Protocol 7.15 provides forward secrecy against eavesdroppers. However, Ding and Horster [104] showed that it is vulnerable to on-line undetectable guessing as shown in Attack 7.1. The insider adversary C records an old protocol run with A and replays A's input in message 2. Only the interaction with S in messages 2 and 3 is relevant for the attack. By guessing π_A the adversary can find the corresponding value $\tilde{t}_A$ that A would have sent and use this value as C's input to message 2. On receipt of message 3, C can check if both returned values are the same in order to confirm a correct guess of π_A.

C has password π_C and recorded message $\{t_A \oplus C\}_{\pi_A}$. C guesses A's password is $\tilde{\pi}$ and chooses $\tilde{t}_A$ such that $\{t_A \oplus C\}_{\pi_A} = \{\tilde{t}_A \oplus C\}_{\tilde{\pi}}$.

2. $C \rightarrow S : A, \{t_A \oplus C\}_{\pi_A}, \{\tilde{t}_A \oplus A\}_{\pi_C}$
3. $S \rightarrow C : t_A^{r_S}, \tilde{t}_A^{r_S}$

C checks if $t_A^{r_S} = \tilde{t}_A^{r_S}$. If so $\tilde{\pi} = \pi_A$.

Attack 7.1: Ding and Horster's attack on Protocol 7.15

Lin *et al.* [207] also discovered an off-line guessing attack on Protocol 7.15, as shown in Attack 7.2. The adversary C masquerades as both A and S in order to find the password of B. The values X and Y are chosen randomly by C and are simply place-holders for missing values. The value r_C is chosen by C in place of $r_A r_S$ in a real protocol run. C knows that B will calculate $Z_{AB} = (g^{r_C})^{r_B} = t_B^{r_C}$ so messages 2 and 4 can be used to check a guess for π_B.

C chooses random values X, Y and r_C.

1. $C_A \rightarrow B : X$
2. $B \rightarrow C_S : A, X, \{t_B \oplus A\}_{\pi_B}$
3. $C_S \rightarrow B : g^{r_C}, Y$
4. $B \rightarrow C_A : Y, \{\text{Message 1}\}_{K_{AB}}$

C guesses $\pi_B = \tilde{\pi}$ and decrypts $\{t_B \oplus A\}_{\pi_B}$ with $\tilde{\pi}$ to obtain $\tilde{t}_B$ and calculates $\tilde{Z}_{AB} = \tilde{t}_B^{r_C}$. Message 4 is used to check if guess is correct.

Attack 7.2: Lin-Sun-Hwang attack on Protocol 7.15

Lin *et al.* [207] proposed an alternative version of Protocol 7.15 which requires the client to know the correct public key of the server. Recently Lin *et al.* [208] have proposed another variant which avoids the known attacks without the need for a server public key.

7.5 RSA-Based Protocols

Although most password-based protocols rely on the Diffie–Hellman key exchange, it is not surprising that the widespread popularity of the RSA algorithm [274] has been the basis of some alternatives. We examine such protocols in this section.

7.5.1 RSA-Based EKE

Bellovin and Merritt's original EKE protocol is applicable to a number of public key algorithms and they included a version for RSA in their paper [38]. The basic model for EKE dictates that A should choose an ephemeral RSA public key and send it to B encrypted with π. B will then choose the session key and return it encrypted with the RSA key and, optionally, with π.

Bellovin and Merritt recognise a number of potential problems with this approach and discuss some possible remedies. The main problem is how to make an RSA key appear 'random' in order to avoid a partition attack which might eliminate candidate passwords that do not provide a valid RSA key pair when applied to decrypt the first message. They point out that sending the RSA key pair encrypted with the password is dangerous since most integer values do not make a good RSA modulus. Therefore a partition attack is possible whereby the adversary decrypts the ciphertext with a candidate password and can eliminate this candidate if the plaintext modulus has small factors. This will allow an adversary to eliminate a large proportion of passwords.

In view of the above problem Bellovin and Merritt propose that only the RSA exponent e be encrypted with π, while the modulus n is sent as cleartext. They further suggested that, since e must always be odd in the RSA algorithm, either e or $e+1$ be sent randomly to ensure that almost all integers less than n could occur as the encrypted value (they suggest to make n the product of two safe primes[3] to enhance this property). The first two messages in the protocol are then as follows.

$$
\begin{array}{ll}
1. & A \rightarrow B : A, n, \{e\}_{\pi} \\
2. & B \rightarrow A : \{K_{AB}^{e} \bmod n\}_{\pi} \\
\ldots &
\end{array}
$$

Notice that an eavesdropper who obtains an old session key K'_{AB} can decrypt the first message with a candidate $\tilde{\pi}$ to obtain a candidate $\tilde{e}$ and then calculate $\{(K'_{AB})^{\tilde{e}} \bmod n\}_{\tilde{\pi}}$. If the RSA encryption is deterministic, this can be used to verify whether $\tilde{\pi} = \pi$ by comparing with the second message in the old protocol run which used K'_{AB}. Therefore it is important that randomness is included with the session key as part of the RSA encryption.

Another necessary measure to prevent partition attacks is to ensure that candidate-decrypted values in the second message are smaller than n; as with

[3] A prime p is a *safe prime* if $(p-1)/2$ is also prime.

Diffie–Hellman-based EKE this may be addressed by ensuring that the modulus is slightly less than a power of 2. However, Patel [262] showed that these precautions are still not sufficient to protect the protocol. The attack of Patel requires the adversary to replace the first message with A, n', X where X is random and n' is carefully chosen so that $n' = pq$ with $3|(p-1)$ and $3|(q-1)$. The choice of n' ensures that the mapping $x \mapsto x^3 \bmod n$ is a 9-to-1 mapping in $\mathbb{Z}_{n'}^*$. On receipt of this message B will 'decrypt' X with π and use the resulting value e' to encrypt the random value K_{AB}. There is a probability of $1/3$ that e' is divisible by 3. Although the adversary cannot know if this is the case, the attack can be repeated if it fails initially. Assuming it is the case, the adversary can partition the candidate passwords into those that decrypt the second message into a cubic residue and those that do not. The latter case occurs with probability about $8/9$ and these candidates can be discarded since the value K_{AB}^e is a cubic residue. Cubic residues can easily be recognised by the adversary by raising to the power $(p-1)/3 \bmod p$ and also raising to the power $(q-1)/3 \bmod q$ and checking that both results equal 1. By repeating the attack with the same first message the adversary can obtain a new reply and use it in the same way to discard a proportion $8/9$ of the remaining candidate passwords. The attack can easily be generalised to use any small value in place of 3 as the divisor. An attempt to avoid the attack by having the receiver check that the decrypted e has no small factors fails because the adversary can now discard passwords that result in such e values.

7.5.2 OKE and SNAPI

Although the attack of Patel seems fatal for RSA-based EKE, a new approach was taken by Lucks [215] to revive the protocol. The protocol of Lucks was called OKE (Open Key Exchange) which emphasises that the public ephemeral RSA key is sent 'in the open' (as plaintext). Lucks provides a proof in a Bellare–Rogaway-style model that OKE is secure in a general setting. However, there are problems in satisfying the assumptions of the proof in the RSA setting; the difficulties revolve around the ability of the adversary to select the public RSA key in such a way as to obtain information about the password, much the same as in Patel's attack outlined above. Indeed MacKenzie *et al.* [219] subsequently did find an attack on the RSA version of OKE along these lines. They also proposed a variant of RSA-based OKE called SNAPI (Secure Network Authentication with Password Information) which avoids the attack and is described in Protocol 7.16.

A critical factor in avoiding the attack on the OKE protocol is the choice of RSA public key. The RSA modulus N is chosen using a security parameter L_1 so that $2^{L_1-2} \leq N \leq 2^{L_1}$. The exponent e must be in the range $2^{L_1} \leq e \leq 2^{L_1+1}$. A consequence is that exponentiation with e is more than a full length exponentiation; with typical values e will be 1025 bits long.

An important check is that the parameter p must lie in the set $S_N = \{x : x \leq 2^\eta - (2^\eta \bmod N) \wedge (x, N) = 1\}$. Since p is an output of H which has

Shared information: Password π. Security parameters L_1 and L_2. A 'long' hash function H with output of length η with $\eta \geq L_1 + L_2$ and three 'short' hash functions H_1, H_2 and H_3 with outputs of length L_2. A set $S_N = \{x : x \leq 2^\eta - (2^\eta \bmod N) \wedge (x, N) = 1\}$.

A		B
$n_A \in_R \{0,1\}^{L_2}$		
Choose RSA public key N, e		
	$\xrightarrow{A, n_A, N, e}$	Check range of N, e
		Check e is prime
		Check length of n_A
		$n_B \in_R \{0,1\}^{L_2}$
		$p = H(N, e, n_A, n_B, A, B, \pi)$
		$a \in_R \mathbb{Z}_N^*$
		If $p \notin S_N$ then $q = a$
		else $q = pa^e \bmod N$
	$\xleftarrow{n_B, q}$	
Check length of n_B		
Check $(q, N) = 1$		
$p = H(N, e, n_A, n_B, A, B, \pi)$		
Check $p \in S_N$		
$a = (q/p)^d \bmod N$		
$r = H_1(N, e, n_A, n_B, A, B, q, a)$	$\xrightarrow{r}$	If $p \notin S_N$ then halt
		Verify r
		$t = H_2(N, e, n_A, n_B, A, B, q, a)$
		$K_{AB} = H_3(N, e, n_A, n_B, A, B, q, a)$
	$\xleftarrow{t}$	
Verify t		
$K_{AB} = H_3(N, e, n_A, n_B, A, B, q, a)$		

Protocol 7.16: SNAPI protocol

length at least $L_1 + L_2$ and N has length not more than L_1, this will fail with tiny probability if N is chosen honestly, but an adversary could choose N with small factors to make the probability high that $(x, N) > 1$. If the condition is not satisfied by the value of p generated then q is set to the random value a so that in any event q appears to A to be random.

MacKenzie *et al.* [219] provide a proof of security of SNAPI in Shoup's simulation model. As well as possessing a security proof, a potential benefit of SNAPI is that the ephemeral RSA key can be reused in several protocol runs. Since RSA key generation is a computationally costly process this is a useful advantage and means that SNAPI can be more efficient than PAK. A

drawback of this approach is that storing the ephemeral key makes it part of the long-term key for A and then compromise of the private key means that old session keys can be recovered for all sessions in which that RSA key was used: in other words, this computational advantage comes at the cost of a loss in forward secrecy. MacKenzie *et al.* [219] also proposed an extension to SNAPI, called SNAPI-X, which allows the server side to store only an image of π under a hash function, similar to protocols in Sect. 7.3.

Another variant of OKE was proposed by Roe *et al.* [276]. A novel feature of their protocol is that A sends only the RSA modulus N while the exponent e is defined as a deterministic function of π; this makes the protocol potentially more efficient than SNAPI (or original OKE). B returns $z^{e(\pi)} \bmod N$ where z is a random number of appropriate length used to define various values including the session key and a nonce n_A. On receipt of this value, A should calculate the decryption key $e(\pi)^{-1} \bmod \phi(N)$ and decrypt z as in normal RSA. Then A returns the value n_A to B. Unfortunately, Bleichenbacher [47] has shown that multiple passwords can be checked by B for each protocol run by sending $z^{e(\pi_1)e(\pi_2)\ldots e(\pi_n)}$ and using the returned n_A value to check which, if any, of the $e(\pi_i)$ values were removed by A.

7.6 Protocols Using a Server Public Key

Some of the earliest proposed password-based protocols assume that users not only share passwords with a server but are also in possession of, or at least can obtain, an authentic server public key. This is a significant additional assumption, but it may be realistic in certain applications; a user may trust a workstation to obtain the server's public key or there may be a means to allow the user to check the public key. Since the user's password must be entrusted to some computing device, it does not seem unreasonable to give such a device some trust in obtaining the server public key. Halevi and Krawczyk [139, 140] have proposed a method whereby users can verify the hash of the server public key using sequences of (English) words.

In contrast to EKE and its variants, password-based protocols employing a server key have no need to use the password as a key for encryption, but instead the password can be encrypted by the user with the server public key. A critical issue with this approach is whether the adversary is able to gain any useful information from the ciphertext containing the password. One immediate objection may be that the adversary can make trial encryptions of candidate passwords and check if the same ciphertext is obtained. Such a possibility illustrates that it is usually necessary for the encryption to be probabilistic so that each time a different ciphertext is obtained. We will make the requirement more precise later.

In this section we include both two-party and three-party protocols. Public key computations on the client side are typically restricted to a single encryption in this class of protocols. The two-party protocols tend to be more

efficient for the client than the two-party protocols using ephemeral public keys. However, in contrast to the EKE-based Diffie–Hellman protocols examined in Sects. 7.2 and 7.3, forward secrecy is often sacrificed.

7.6.1 GLNS Protocols with Server Public Keys

In addition to their 'secret public key' protocol examined in Sect. 7.4.1, Gong *et al.* [130] published several protocols assuming server public keys are available to clients. We examine some of these and subsequent variants from other authors. Unfortunately none of these protocols carries a security proof.

Protocol 7.17 shows the 'compact' version of the GLNS protocol for establishing a new session key between A and B who initially share passwords π_A and π_B with the server S. There is a strong similarity with messages 3 to 7 of Protocol 7.12.

S has passwords π_A and π_B.
A has password π_A. A chooses random values n_A, n'_A, c_A, r_A and timestamp T_A.
B has password π_B. B chooses random values n_B, n'_B, c_B, r_B and timestamp T_B.

1. $A \rightarrow B : E_S(A, B, n_A, n'_A, c_A, \{T_A\}_{\pi_A}), r_A$
2. $B \rightarrow S : E_S(A, B, n_A, n'_A, c_A, \{T_A\}_{\pi_A}), E_S(B, A, n_B, n'_B, c_B, \{T_B\}_{\pi_B})$
3. $S \rightarrow B : \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{n_B, K_{AB} \oplus n'_B\}_{\pi_B}$
4. $B \rightarrow A : \{n_A, K_{AB} \oplus n'_A\}_{\pi_A}, \{H_1(r_A), r_B\}_{K_{AB}}$
5. $A \rightarrow B : \{H_2(r_B)\}_{K_{AB}}$

Protocol 7.17: GLNS compact protocol

As in Protocol 7.12, the 'confounders' c_A and c_B may be omitted by requiring that the asymmetric encryption algorithm provides semantic security. Also the asymmetric encryption algorithm must provide non-malleability. If the long-term decryption key of S is compromised then n_A and n'_A can be found, allowing a brute force search for π_A, and consequently revealing the session key. Therefore we conclude that forward secrecy is not provided.

The encrypted timestamp, $\{T_A\}_{\pi_A}$, included in the first message, is used by S to ensure that the message is freshly generated by A. Without this timestamp the adversary could replay message 1 and obtain two messages $\{n_A, K_{AB} \oplus n'_A\}_{\pi_A}$ and $\{n_A, K'_{AB} \oplus n'_A\}_{\pi_A}$, the only difference being in the encrypted session keys. Then the adversary could again mount a brute force attack on π_A since a correct guess can be identified when the first components of the two decrypted messages are identical. (This attack was observed by Tsudik and van Herreweghen [321] as well as by Gong *et al.* [130].) The use of timestamps requires the server to record all messages received for a period of the maximum time window allowed for clock differences and message delay.

In order to remove this drawback Gong *et al.* also provided a version of the protocol in which the server generates a nonce that must be sent with the client requests; its drawback in turn is the addition of two messages to the protocol. Tsudik and van Herreweghen [321], and later Gong [129], have proposed a variant of this nonce-based protocol, aimed at simplifying it and reducing computational requirements. Protocol 7.18 shows Gong's optimal version, in which the number of messages exchanged is reduced to five. Gong [129] observes that this is the same number of messages used in the timestamp-based version.

S has passwords π_A and π_B.
A has password π_A. A chooses random values $n_A, n'_A, n''_A, c_A, r_A$.
B has password π_B. B chooses random values $n_B, n'_B, n''_B, c_B, r_B$.

1. $A \rightarrow B : E_S(A, B, n_A, n'_A, c_A, n''_A, \{n''_A\}_{\pi_A}), r_A$
2. $B \rightarrow S : E_S(A, B, n_A, n'_A, c_A, n''_A, \{n''_A\}_{\pi_A}),$
   $E_S(B, A, n_B, n'_B, c_B, n''_B, \{n''_B\}_{\pi_B})$
3. $S \rightarrow B : \{n_A, K_{AB} \oplus n'_A\}_{n''_A}, \{n_B, K_{AB} \oplus n'_B\}_{n''_B}$
4. $B \rightarrow A : \{n_A, K_{AB} \oplus n'_A\}_{n''_A}, \{H_1(r_A), r_B\}_{K_{AB}}$
5. $A \rightarrow B : \{H_2(r_B)\}_{K_{AB}}$

Protocol 7.18: Optimal GLNS nonce-based protocol

The main difference between the nonce-based and timestamp-based protocols is that in the former A and B send a third nonce to S, which is used both to authenticate them to S and also as a shared secret to encrypt the session key in the reply from S.

7.6.2 Kwon–Song Protocols

Kwon and Song [196, 197] proposed a set of password-based protocols. Protocol 7.19 is their basic protocol. The session key is generated by B and masked using nonces sent by A in the first message. The protocol is simple and the computation required for both parties is small. A drawback is that this protocol does not provide forward secrecy.

The final message in Protocol 7.19 prevents a replay attack. An adversary who obtains an old K_{AB} value can replay the first message and obtain the new session key since B will reuse the same r_A and r'_A values. However, the adversary cannot complete the protocol since r_A and r'_A are still unknown so the correct final message cannot be formed.

Kwon and Song provide a GNY logic analysis of their basic protocol. However, there is no specification of the requirements for the public key encryption algorithm. There does not seem to be any specific requirement for

Shared information: Hash function H. Password π. Authentic public key of B.

A		B
Choose random r_A, r'_A		
	$\xrightarrow{E_B(r_A, r'_A, \pi \oplus r'_A)}$	Decrypt and verify
		Choose K_{AB}
		$C = K_{AB} \oplus r_A \oplus r'_A$
Decrypt C and verify hash	$\xleftarrow{C, H(\pi \oplus r_A, K_{AB}, r'_A)}$	
	$\xrightarrow{H(\pi \oplus r'_A, K_{AB}, r_A)}$	Verify hash

Protocol 7.19: Kwon–Song basic protocol

non-malleability since an adversary should not be able to obtain any of the encrypted values, even if an old session key is obtained.

Kwon and Song [196, 197] also provide some variant protocols. Their 'Challenger's Public Key Protocol' requires the responder to know the authentic public key of the initiator but includes an unspecified symmetric encryption algorithm using the password. They also present two key agreement protocols, the first of which does not provide forward secrecy while the second is essentially the same as Diffie–Hellman-based EKE.

7.6.3 Halevi–Krawczyk Protocols

Halevi and Krawczyk [139, 140] designed password-based protocols in a modular way and conducted a formal analysis. They provided a formal proof of security for the challenge–response protocol that forms the basis of their other protocols, but only an outline argument of how the proof can be extended to key establishment. Boyarsky [53] argued that the protocols in the original paper [139] cannot be secure without a stronger assumption on the asymmetric encryption algorithm used unless only one user is involved. A stronger assumption is used in the later paper of Halevi and Krawczyk [140] but still not as strong as suggested by Boyarsky. However, all parties agree that a cryptosystem that provides non-malleability is sufficient for security.

In all their protocols Halevi and Krawczyk assume that the user A already has possession of a 'public password' which is a hashed version of the public key of the server B. This value can be used to verify the server's full public key which is sent to the user as part of the protocol. We omit this in our descriptions. Protocols for entity authentication only are proposed in addition to protocols for key exchange with and without forward secrecy.

The key establishment protocols use a similar design to the SKEME protocols defined in Protocols 5.21 and 5.38 in Chap. 5. We present the version

without forward secrecy in Protocol 7.20. In order to preserve our usual convention that A is the client and B the server, the protocol is shown with the first message originating with B. In practice A would often need to start the protocol with a request to B to connect, which would precede the protocol as shown.

Shared information: Password π. Security parameter L.

A		B
		$r_B \in_R \{0,1\}^L$
Choose random K_0	$\xleftarrow{r_B}$	
	$\xrightarrow{E_B(K_0, \pi, r_B, A, B)}$	Decrypt and verify
		$y = MAC_{K_0}(r_B, B, A)$
Verify y	$\xleftarrow{y}$	$K_{AB} = MAC_{K_0}(y)$
$K_{AB} = MAC_{K_0}(y)$		

Protocol 7.20: Halevi–Krawczyk password-based protocol

Protocol 7.20 is a simple version of the protocol of Halevi and Krawczyk. More generally the encrypted message sent from A to B can be of the form $E_B(K_0, f_\pi(r_B, K_0, A, B))$ for a function $f_\pi(X)$ used to combine the password with the other protocol fields requiring verification. It is required that f is one-to-one (injective) when either π or X is fixed. Therefore a suitable instantiation of $f_\pi(X)$ is the concatenation of π and X as used in Protocol 7.20. Note that although K_0 is transported from A to B, the session key K_{AB} is formed by key agreement since it includes inputs from both parties.

The properties and computational requirements are similar to Protocol 7.19 but a difference is that in Protocol 7.20 the server and client both contribute to the session key, whereas in Protocol 7.19 it is the server that generates the key. Like the SKEME variant Protocol 5.38, Protocol 7.20 does not provide forward secrecy. Another protocol with the same basic design as Protocol 7.20, but incorporating a Diffie–Hellman exchange, was proposed by Halevi and Krawczyk to provide this property. As usual there is a computational cost in so doing, which amounts to two extra exponentiations per user.

7.6.4 Three-Party Protocol of Yen and Liu

The protocols of Yen and Liu [339] use ideas from the simplified GLNS three-party EKE of Tsudik and van Herreweghen (Protocol 7.13). By making use of a server public key they avoid the need for ephemeral public keys. Protocol

7.21 shows their main protocol, in which the server generates the session key K_{AB}. They also propose variants in which either the initiator or responder can generate K_{AB}.

S has passwords π_A and π_B. S chooses K_{AB}.
A has password π_A. A chooses random values n_A, n'_A.
B has password π_B. B chooses random value n_B.

1. $A \rightarrow B : A, n'_A, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A)$
2. $B \rightarrow S : A, B, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A), E_S(n_B \oplus \pi_B, n_B \oplus A \oplus n'_A)$
3. $S \rightarrow A : n_A \oplus \{n_A \oplus K_{AB}\}_{\pi_A}, \{n'_A\}_{K_{AB}}, n_B \oplus \{n_B \oplus K_{AB}\}_{\pi_B}$
4. $A \rightarrow B : n_B \oplus \{n_B \oplus K_{AB}\}_{\pi_B}, \{n'_A + 1\}_{K_{AB}}$

Protocol 7.21: Yen–Liu protocol

On receipt of message 2, S decrypts both ciphertexts to obtain two pairs of values, say X_A, Y_A and X_B, Y_B. Then S sets $n_A = X_A \oplus \pi_A$ and $n_B = X_B \oplus \pi_B$ and checks that $n_A \oplus Y_A \oplus B = n_B \oplus Y_B \oplus A$. If not, then S will abort the protocol. Despite a detailed analysis by its authors, the Yen–Liu protocol does not provide authentication of both parties. An insider adversary C is able to masquerade as B and successfully complete a protocol run with A, including obtaining the new session key.

---

1. $A \rightarrow C_B : A, n'_A, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A)$
2. $C \rightarrow S : A, C, E_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A), E_S(n_C \oplus \pi_C, n_C \oplus A \oplus n''_A)$
3. $S \rightarrow C_A : n_A \oplus \{n_A \oplus K_{AC}\}_{\pi_A}, \{n''_A\}_{K_{AC}}, n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}$

3′. $C_S \rightarrow A : n_A \oplus \{n_A \oplus K_{AC}\}_{\pi_A}, \{n'_A\}_{K_{AC}}, n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}$

4. $A \rightarrow C_B : n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}, \{n'_A + 1\}_{K_{AC}}$

Attack 7.3: Attack on Protocol 7.21

In Attack 7.3, A wishes to complete the protocol with B, but in fact completes it with the adversary C. After receiving message 1 from A, C generates a new value n''_A such that $n''_A \oplus C = n'_A \oplus B$. This enables C to send a correct message 2 to S as if A is intending to run the protocol with C. C needs to intercept message 3 from S in order to replace the field $\{n''_A\}_{K_{AC}}$ with the field $\{n'_A\}_{K_{AC}}$ expected by A. This can be done since C is able to extract K_{AC} from message 3. When A receives the altered message 3′ the session key K_{AC} will be extracted by A and used to confirm that the value n'_A was correctly received. Thus A will accept the key as shared with B whereas it is actually shared with C.

7.7 Other Protocols

This section contains some protocols that do not fit into the pattern of the main trends of protocols.

7.7.1 Lee–Sohn–Yang–Won Protocol

The protocol of Lee *et al.* [202] is based on the Nyberg–Rueppel protocol which was examined in Chap. 5 (Protocol 5.3). Protocol 7.22 consists of basic Diffie–Hellman key agreement with a form of signature using the password as key; an image of the password (corresponding to a public key in the Nyberg–Rueppel protocol) is used for verification by the server. In addition, the exchanged hashes are intended to provide key confirmation.

Shared information: Hash functions H_1 and H_2. Element g generating group G of order q.
Information held by A: Password π.
Information held by B: Password image $P = g^{-H_2(\pi)}$.

A		B
$r_A, s_A \in_R \mathbf{Z}_q$ $e = g^{r_A - s_A}$		
$f = s_A + eH_2(\pi) \bmod q$	$\xrightarrow{A, e, f}$	$t_A = P^e g^f e$ $r_B \in_R \mathbf{Z}_q$ $Z_{AB} = t_A^{r_B}$ $t_B = g^{r_B}$
$Z_{AB} = t_B^{r_A}$ Verify hash	$\xleftarrow{t_B, H_1(Z_{AB}, e, f)}$	
	$\xrightarrow{H_1(Z_{AB}, t_B)}$	Verify hash

Protocol 7.22: Lee–Sohn–Yang–Won protocol

Unfortunately this protocol can be broken by an active adversary. The adversary C chooses r_C randomly, calculates $e = g^{r_C}$ and chooses f at random. The adversary C masquerades as A and sends A, e, f as the first message. Now C knows that B will calculate the shared secret as:

$$Z_{AB} = (P^e g^{f + r_C})^{r_B} = t_B^{-H_2(\pi)e + f + r_C}$$

and the only element unknown to C here is π. Therefore, on receipt of $t_B, H_1(Z_{AB}, e, f)$ from B, the adversary can guess the password as $\tilde{\pi}$ and

check if this is the correct π by calculating the corresponding Z_{AB} value and checking against the hash. This process allows C to engage in repeated password guessing until the correct password of A is found.

7.7.2 Anderson–Lomas Protocol

We will now describe an ingenious proposal of Anderson and Lomas [13]. Although it does not turn out as effective as many of the alternative protocols it is worth examining because of its originality. The novelty is the use of a *collisionful* hash function. In contrast to the usual requirement for hash functions these are functions for which it is *easy* to find many collisions for any input. Such functions are described by Gong [128] and we adopt his definition here.

Definition 7.1. *A* collisionful hash function *is a function q that maps a key k and a bit-string x and satisfies:*

- *Given k and x it is easy to compute $q_k(x)$.*
- *Given k and $q_k(x)$ it is hard to find a value y such that $q_k(x) = q_k(y)$ but $x \neq y$.*
- *Given pairs of values x and $q_k(x)$ it is hard to compute k, though it is less hard to find k' such that $q_k(x) = q_{k'}(x)$ but $k \neq k'$.*

The protocol of Anderson and Lomas uses a specific collisionful function:

$$q_k(x) = H(MAC_k(x) \bmod 2^m, x)$$

where m is a parameter and H is a one-way hash function. Since only m bits of the function depend on k it should be easy to find many k values that give the same output when m is small. Anderson and Lomas suggest to take $m = n/2$ where 2^n is the size of the password space. The idea of the protocol is then to use π as the key for q and so a guess for π can only be verified with probability $2^{-n/2}$.

A and B share password π.
A chooses random value r_A and sets $t_A = g^{r_A}$.
B chooses random value r_B and sets $t_B = g^{r_B}$.

1. $A \rightarrow B : t_A$
2. $B \rightarrow A : t_B$
3. $A \rightarrow B : q_\pi(Z_{AB})$
4. $B \rightarrow A : q_{H(\pi)}(Z_{AB})$

Protocol 7.23: Anderson–Lomas protocol

Protocol 7.23 shows the Anderson–Lomas protocol which is simply the basic Diffie–Hellman exchange followed by an exchange of authenticating values using function q. The shared secret Z_{AB} is the Diffie–Hellman ephemeral secret, $g^{r_A r_B}$. An adversary can try to mount an active attack by posing as B and attempting to guess the password from message 3; this will succeed with probability only $2^{-n/2}$.

An active adversary can reduce the number of possible passwords to the square root of the initial number. Thus the protocol leaks much more information than is desirable. Anderson and Lomas state that if A detects a failed login then she must change her password, which is somewhat inconvenient. Bakhtiari *et al.* [23] have pointed out that if Z_{AB} becomes known to the adversary then an off-line guessing attack becomes possible.

7.7.3 Strengthening Passwords

We mention here some interesting ideas that are aimed at 'strengthening' passwords. These techniques may be regarded as alternatives to using password-based authentication since they allow a password known by a user to be converted into a better quality shared secret. Each of these proposals, however, seems to have some significant limitations if a high level of security is required.

- Abadi *et al.* [3] suggested an ingenious scheme for randomising a password π shared between a user and server. At authentication time the user chooses a random value ρ of, say, 20 bits and sends the value $H(\pi, \rho)$, using a one-way hash function H. The server then tries all 2^{20} possible ρ values together with the known value of π and when the correct one is found the pair (π, ρ) may be used as a longer shared secret to form a session key. The adversary who eavesdrops and attempts a dictionary attack on the combination of (π, ρ) finds that the effort is 2^{20} times as hard as just to guess π. Although the extra computational effort required by the server may be acceptable in some scenarios, it places a limit on the level of security that can be obtained by this method.
- Perlman and Kaufman [268] explored 'bootstrapping' of a public key infrastructure on a workstation with no stored user information. The user's public and private keys, as well as any other necessary information, are stored on a server together with the user's password. By setting up a secure session with the server, this information can be safely downloaded from the server.
- Ford and Kaliski [113] proposed a method of *password hardening* which allows a user A to obtain a strong secret from her password π in cooperation with a server S. The server does not know the user's password but must have a user-specific exponent d_A which is used in a multiplicative group. (We assume here that π is an element in the group and is of high order, but in practice a suitable mapping from the string representing π

into the group may be necessary.) Each time A wishes to recover her hardened password she generates a random k and sends π^k to S. The server S returns π^{kd_A} and then A removes k by calculating $(\pi^{kd_A})^{k^{-1}}$ to obtain the strengthened password π^{d_A}. Notice that A recovers the same long secret each time. Ford and Kaliski discuss how to use this technique with multiple password servers to provide a secure password-based authentication service. One useful feature is that compromise of a subset of these servers does not compromise the password. Jablon [166] pointed out that an authenticated channel is required between the user and servers to prevent the adversary mounting a password guessing attack. He proposes an enhanced protocol to deal with this problem.

7.8 Conclusion

Password-based protocols allow users to establish a strong shared session key with other principals using no secret other than a short string that can be committed to human memory. Considering their more stringent requirements, it may be expected that such protocols are harder to design than authentication and key establishment protocols with full-length keys. However, understanding of password-based protocols has advanced rapidly in little more than 10 years since it was first realised that they are possible at all. Today there are several examples that have security assurance and practical performance similar to what can be achieved for protocols using full-length keys.

Some of the properties of the most prominent protocols examined in this chapter are summarised in Table 7.2. Notice that it is possible to run the SPEKE protocol without authenticators so that it uses only two messages like PPK. It is worth repeating that all the security proofs for protocols indicated in Table 7.2 rely on the random oracle model, except for the Katz–Ostrovsky–Yung (KOY) protocol.

The number of messages is one measure of the communications efficiency that is easy to compare in the different protocols. The computational efficiency is much harder to compare because most authors are not explicit about the size of parameters to be used. Furthermore, when security is based on different computational problems the relationship between security and parameter size can be quite different too.

The public key operations dominate the other computations in all the protocols in this chapter and so we only count these. Table 7.3 attempts to compare the performance of the class of EKE and augmented EKE protocols from the client side. It is unfortunate that in many protocols the client side, which may be much more limited in computational ability, has higher computational requirements than the server side. This is because the client usually needs to calculate an image of the password, whereas this can be stored directly by the server. An exception to this rule is PAK-R which can also be combined with other PAK variants.

Table 7.2. Properties of password-based protocols

	No. of messages	Security proof	Server holds image of π	Server PK required
DH-EKE (7.1)	4	No	No	No
PAK (7.2)	3	Yes	No	No
PPK (7.3)	2	Yes	No	No
PAK-R (7.4)	3	Yes	No	No
SPEKE (7.5)	3	Yes	No	No
KOY (7.6)	3	Yes	No	No
Augmented EKE (7.7)	5	Broken	Yes	No
PAK-Y (7.8)	3	Yes	Yes	No
B-SPEKE (7.9)	3	No	Yes	No
SRP (7.10)	4	No	Yes	No
AMP (7.11)	4	No	Yes	No
OKE	4	Broken	No	No
SNAPI (7.16)	4	Yes	No	No
SNAPI-X	4	Yes	Yes	No
GLNS (7.17)	5	No	No	Yes
Halevi–Krawczyk (7.20)	3	Partial	No	Yes

In Table 7.3 we assume that modulus sizes are chosen at 1024 bits for either Diffie–Hellman or RSA protocols. For the Diffie–Hellman-based protocols the basic requirements for each principal are two exponentiations: one to calculate t_A and the other to calculate Z_{AB}. Although this minimum is apparently achieved in the original Diffie–Hellman EKE its vulnerabilities make it a dubious choice today. For B-SPEKE we have assumed that the password size is 32 bits. Also for SRP we have assumed that the small hash $H(s, \pi)$ yields a 32-bit value. The requirements for each protocol are estimated by counting the number of public key operations; it must be appreciated that this is only an indication of the relative computational effort which depends on several detailed factors. For the Diffie–Hellman-based protocols the public key operations are taken as the exponentiations required for each side. It is perhaps remarkable that today it seems that the best password-based protocols enjoy most of the good properties that we expect of protocols using long secrets.

When selecting a protocol for use, undoubtedly the most important factor is its security. Since there are now a number of protocols with proven security, and especially bearing in mind the many subtle attacks found on earlier protocols, it seems prudent to use one of these. Examination of Tables 7.2 and 7.3

Table 7.3. Client side computation in password-based protocols

	No. of exponentiations of this size			
	32 bits	160 bits	864 bits	1024 bits
EKE				
DH-EKE		2		
PAK		2	1	
PPK		2	2	
PAK-R		3		
SPEKE		2		
SNAPI				1[†]
Augmented EKE				
Augmented EKE		2	2	
PAK-Y		3	1	
B-SPEKE	1	2		
SRP	2	2		
AMP		2		

[†] With additional checks an exponent as small as 512 bits is possible [219].

reveals that for EKE using Diffie–Hellman or RSA we can select a protocol with a formal proof of security with little sacrifice in performance. However, for the class of augmented EKE protocols there is currently a significant gap between the most efficient protocols with a formal security proof and those without one.

Of those protocols based on discrete logarithms that have a security proof, PAK and SPEKE are the most efficient. Options to exclude explicit authentication and to store only an image of the password at the server (PAK-Y and B-SPEKE) make these flexible choices too. The client side computation can be reduced in PAK-Y by the same trick as PAK-R; this replaces the 864-bit exponentiation with an additional 160-bit exponentiation. The only reasonable choice for RSA-based protocols is SNAPI. Although SNAPI can be more efficient than PAK if a fixed RSA modulus is used, the loss of forward secrecy may be a significant price to pay in many applications, while the increased number of messages may be an important drawback too. The emerging IEEE P1363.2 standard [155] includes detailed specifications of all the protocols in Table 7.3 in various different algebraic settings.

The GLNS protocols provide a means for two users to establish a session key with each other when they both share only weak keys with a server. These protocols, together with Protocol 7.15 and its variants, seem to be the only

ones that have been published for this purpose. For client–server protocols where the user possesses (or can obtain) an authentic copy of the server's public key, the protocols of Halevi and Krawczyk are efficient and enjoy a proof of security as long as Boyarsky's advice, to use a public key encryption algorithm secure against adaptive chosen ciphertext attacks, is followed.

As in Chap. 5, we have described all the protocols in this chapter in the context of subgroups of $\mathbb{Z}_p^*$, but many of them can be generalised to a variety of different groups. Since these protocols may well be useful in applications employing mobile computing devices, the computational efficiency and storage gains in using elliptic curve groups can be very attractive. Although it is straightforward to generalise the protocol definitions to different groups, there may be undesirable consequences with respect to security. For example, consider Protocol 7.1 when the Diffie–Hellman exchange takes place in an elliptic curve group. Encryption of an elliptic curve point using a password is more tricky than encrypting an element of $\mathbb{Z}_p^*$ with its natural mapping to a bit-string. However, protocols where the symmetric encryption algorithm is matched to the group used, such as Protocol 7.2, do not seem to have this problem as long as a suitable means to map the password to an elliptic curve point is available. The emerging IEEE P1363.2 standard [155] has addressed this issue.