Chapter 10

Identification and Entity Authentication

Contents in Brief

10.1 Introduction

This chapter considers techniques designed to allow one party (the *verifier*) to gain assurances that the identity of another (the *claimant*) is as declared, thereby preventing impersonation. The most common technique is by the verifier checking the correctness of a message (possibly in response to an earlier message) which demonstrates that the claimant is in possession of a secret associated by design with the genuine party. Names for such techniques include *identification*, *entity authentication*, and (less frequently) *identity verification*. Related topics addressed elsewhere include message authentication (data origin authentication) by symmetric techniques (Chapter 9) and digital signatures (Chapter 11), and authenticated key establishment (Chapter 12).

A major difference between entity authentication and message authentication (as provided by digital signatures or MACs) is that message authentication itself provides no timeliness guarantees with respect to when a message was created, whereas entity authentication involves corroboration of a claimant's identity through actual communications with an associated verifier during execution of the protocol itself (i.e., in *real-time*, while the verifying entity awaits). Conversely, entity authentication typically involves no meaningful message other than the claim of being a particular entity, whereas message authentication does. Techniques which provide both entity authentication and key establishment are deferred to Chapter 12; in some cases, key establishment is essentially message authentication where the message is the key.

Chapter outline

The remainder of §10.1 provides introductory material. §10.2 discusses identification schemes involving fixed passwords including Personal Identification Numbers (PINs), and providing so-called weak authentication; one-time password schemes are also considered. §10.3 considers techniques providing so-called strong authentication, including challenge-response protocols based on both symmetric and public-key techniques. It includes discussion of time-variant parameters (TVPs), which may be used in entity authentication protocols and to provide uniqueness or timeliness guarantees in message authentication. §10.4 examines customized identification protocols based on or motivated by zero-knowledge techniques. §10.5 considers attacks on identification protocols. §10.6 provides references and further chapter notes.

10.1.1 Identification objectives and applications

The general setting for an identification protocol involves a *prover* or *claimant* A and a *verifier* B. The verifier is presented with, or presumes beforehand, the purported identity of the claimant. The goal is to corroborate that the identity of the claimant is indeed A, i.e., to provide entity authentication.

10.1 Definition *Entity authentication* is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired).

10.2 Remark (*identification terminology*) The terms *identification* and *entity authentication* are used synonymously throughout this book. Distinction is made between weak, strong, and zero-knowledge based authentication. Elsewhere in the literature, sometimes identification implies only a claimed or stated identity whereas entity authentication suggests a corroborated identity.

(i) Objectives of identification protocols

From the point of view of the verifier, the outcome of an entity authentication protocol is either *acceptance* of the claimant's identity as authentic (completion with acceptance), or *termination without acceptance* (rejection). More specifically, the objectives of an identification protocol include the following.

1. In the case of honest parties A and B, A is able to successfully authenticate itself to B, i.e., B will complete the protocol having accepted A's identity.
2. (*transferability*) B cannot reuse an identification exchange with A so as to successfully impersonate A to a third party C.
3. (*impersonation*) The probability is negligible that any party C distinct from A, carrying out the protocol and playing the role of A, can cause B to complete and accept A's identity. Here *negligible* typically means "is so small that it is not of practical significance"; the precise definition depends on the application.
4. The previous points remain true even if: a (polynomially) large number of previous authentications between A and B have been observed; the adversary C has participated in previous protocol executions with either or both A and B; and multiple instances of the protocol, possibly initiated by C, may be run simultaneously.

The idea of zero-knowledge-based protocols is that protocol executions do not even reveal any partial information which makes C's task any easier whatsoever.

An identification (or entity authentication) protocol is a "real-time" process in the sense that it provides an assurance that the party being authenticated is operational at the time of protocol execution – that party is taking part, having carried out some action since the start of the protocol execution. Identification protocols provide assurances only at the particular instant in time of successful protocol completion. If ongoing assurances are required, additional measures may be necessary; see §10.5.

(ii) Basis of identification

Entity authentication techniques may be divided into three main categories, depending on which of the following the security is based:

1. *something known*. Examples include standard passwords (sometimes used to derive a symmetric key), Personal Identification Numbers (PINs), and the secret or private keys whose knowledge is demonstrated in challenge-response protocols.
2. *something possessed*. This is typically a physical accessory, resembling a passport in function. Examples include magnetic-striped cards, *chipcards* (plastic cards the size of credit cards, containing an embedded microprocessor or integrated circuit; also called *smart cards* or *IC cards*), and hand-held customized calculators (*password generators*) which provide time-variant passwords.
3. *something inherent* (to a human individual). This category includes methods which make use of human physical characteristics and involuntary actions (*biometrics*), such as handwritten signatures, fingerprints, voice, retinal patterns, hand geometries, and dynamic keyboarding characteristics. These techniques are typically non-cryptographic and are not discussed further here.

(iii) Applications of identification protocols

One of the primary purposes of identification is to facilitate access control to a resource, when an access privilege is linked to a particular identity (e.g., local or remote access to computer accounts; withdrawals from automated cash dispensers; communications permissions through a communications port; access to software applications; physical entry to restricted areas or border crossings). A password scheme used to allow access to a user's computer account may be viewed as the simplest instance of an *access control matrix*: each resource has a list of identities associated with it (e.g., a computer account which authorized entities may access), and successful corroboration of an identity allows access to the authorized resources as listed for that entity. In many applications (e.g., cellular telephony) the motivation for identification is to allow resource usage to be tracked to identified entities, to facilitate appropriate billing. Identification is also typically an inherent requirement in authenticated key establishment protocols (see Chapter 12).

10.1.2 Properties of identification protocols

Identification protocols may have many properties. Properties of interest to users include:

1. *reciprocity of identification*. Either one or both parties may corroborate their identities to the other, providing, respectively, *unilateral* or *mutual* identification. Some techniques, such as fixed-password schemes, may be susceptible to an entity posing as a verifier simply in order to capture a claimant's password.
2. *computational efficiency*. The number of operations required to execute a protocol.

3. *communication efficiency*. This includes the number of passes (message exchanges) and the bandwidth required (total number of bits transmitted).
   More subtle properties include:
4. *real-time involvement of a third party (if any)*. Examples of third parties include an on-line *trusted* third party to distribute common symmetric keys to communicating entities for authentication purposes; and an on-line (untrusted) directory service for distributing public-key certificates, supported by an off-line certification authority (see Chapter 13).
5. *nature of trust required in a third party (if any)*. Examples include trusting a third party to correctly authenticate and bind an entity's name to a public key; and trusting a third party with knowledge of an entity's private key.
6. *nature of security guarantees*. Examples include provable security and zero-knowledge properties (see §10.4.1).
7. *storage of secrets*. This includes the location and method used (e.g., software only, local disks, hardware tokens, etc.) to store critical keying material.

Relation between identification and signature schemes

Identification schemes are closely related to, but simpler than, digital signature schemes, which involve a variable message and typically provide a non-repudiation feature allowing disputes to be resolved by judges after the fact. For identification schemes, the semantics of the message are essentially fixed – a claimed identity at the current instant in time. The claim is either corroborated or rejected immediately, with associated privileges or access either granted or denied in real time. Identifications do not have "lifetimes" as signatures do[1] – disputes need not typically be resolved afterwards regarding a prior identification, and attacks which may become feasible in the future do not affect the validity of a prior identification. In some cases, identification schemes may also be converted to signature schemes using a standard technique (see Note 10.30).

10.2 Passwords (weak authentication)

Conventional password schemes involve time-invariant passwords, which provide so-called *weak authentication*. The basic idea is as follows. A *password*, associated with each user (entity), is typically a string of 6 to 10 or more characters the user is capable of committing to memory. This serves as a shared secret between the user and system. (Conventional password schemes thus fall under the category of symmetric-key techniques providing unilateral authentication.) To gain access to a system resource (e.g., computer account, printer, or software application), the user enters a (userid, password) pair, and explicitly or implicitly specifies a resource; here *userid* is a claim of identity, and *password* is the evidence supporting the claim. The system checks that the password matches corresponding data it holds for that userid, and that the stated identity is authorized to access the resource. Demonstration of knowledge of this secret (by revealing the password itself) is accepted by the system as corroboration of the entity's identity.

Various password schemes are distinguished by the means by which information allowing password verification is stored within the system, and the method of verification. The collection of ideas presented in the following sections motivate the design decisions

[1] Some identification techniques involve, as a by-product, the granting of *tickets* which provide time-limited access to specified resources (see Chapter 13).

made in typical password schemes. A subsequent section summarizes the standard attacks these designs counteract. Threats which must be guarded against include: password disclosure (outside of the system) and line eavesdropping (within the system), both of which allow subsequent replay; and password guessing, including dictionary attacks.

10.2.1 Fixed password schemes: techniques

(i) Stored password files

The most obvious approach is for the system to store user passwords cleartext in a system password file, which is both read- and write-protected (e.g., via operating system access control privileges). Upon password entry by a user, the system compares the entered password to the password file entry for the corresponding userid; employing no secret keys or cryptographic primitives such as encryption, this is classified as a non-cryptographic technique. A drawback of this method is that it provides no protection against privileged insiders or *superusers* (special userids which have full access privileges to system files and resources). Storage of the password file on backup media is also a security concern, since the file contains cleartext passwords.

(ii) "Encrypted" password files

Rather than storing a cleartext user password in a (read- and write-protected) password file, a one-way function of each user password is stored in place of the password itself (see Figure 10.1). To verify a user-entered password, the system computes the one-way function of the entered password, and compares this to the stored entry for the stated userid. To preclude attacks suggested in the preceding paragraph, the password file need now only be write-protected.

10.3 Remark (*one-way function vs. encryption*) For the purpose of protecting password files, the use of a one-way function is generally preferable to reversible encryption; reasons include those related to export restrictions, and the need for keying material. However, in both cases, for historical reasons, the resulting values are typically referred to as "encrypted" passwords. Protecting passwords by either method before transmission over public communications lines addresses the threat of compromise of the password itself, but alone does not preclude disclosure or replay of the transmission (cf. Protocol 10.6).

(iii) Password rules

Since dictionary attacks (see §10.2.2(iii)) are successful against predictable passwords, some systems impose "password rules" to discourage or prevent users from using "weak" passwords. Typical password rules include a lower bound on the password length (e.g., 8 or 12 characters); a requirement for each password to contain at least one character from each of a set of categories (e.g., uppercase, numeric, non-alphanumeric); or checks that candidate passwords are not found in on-line or available dictionaries, and are not composed of account-related information such as userids or substrings thereof.

Knowing which rules are in effect, an adversary may use a modified dictionary attack strategy taking into account the rules, and targeting the weakest form of passwords which nonetheless satisfy the rules. The objective of password rules is to increase the entropy (rather than just the length) of user passwords beyond the reach of dictionary and exhaustive search attacks. *Entropy* here refers to the uncertainty in a password (cf. §2.2.1); if all passwords are equally probable, then the entropy is maximal and equals the base-2 logarithm of the number of possible passwords.

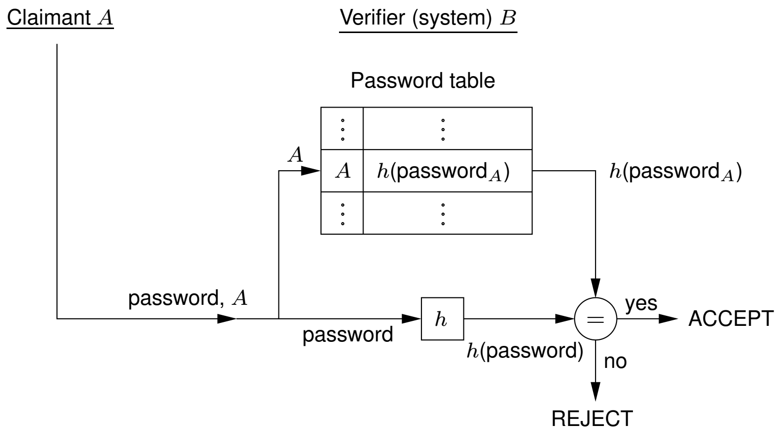

Figure 10.1: *Use of one-way function for password-checking.*

Another procedural technique intended to improve password security is *password aging*. A time period is defined limiting the lifetime of each particular password (e.g., 30 or 90 days). This requires that passwords be changed periodically.

(iv) Slowing down the password mapping

To slow down attacks which involve testing a large number of trial passwords (see §10.2.2), the password verification function (e.g., one-way function) may be made more computationally intensive, for example, by iterating a simpler function $t > 1$ times, with the output of iteration i used as the input for iteration $i + 1$. The total number of iterations must be restricted so as not to impose a noticeable or unreasonable delay for legitimate users. Also, the iterated function should be such that the iterated mapping does not result in a final range space whose entropy is significantly decimated.

(v) Salting passwords

To make dictionary attacks less effective, each password, upon initial entry, may be augmented with a t-bit random string called a *salt* (it alters the "flavor" of the password; cf. §10.2.3) before applying the one-way function. Both the hashed password and the salt are recorded in the password file. When the user subsequently enters a password, the system looks up the salt, and applies the one-way function to the entered password, as altered or augmented by the salt. The difficulty of exhaustive search on any particular user's password is unchanged by salting (since the salt is given in cleartext in the password file); however, salting increases the complexity of a dictionary attack against a large set of passwords simultaneously, by requiring the dictionary to contain 2^t variations of each trial password, implying a larger memory requirement for storing an encrypted dictionary, and correspondingly more time for its preparation. Note that with salting, two users who choose the same password have different entries in the system password file. In some systems, it may be appropriate to use an entity's userid itself as salt.

(vi) Passphrases

To allow greater entropy without stepping beyond the memory capacity of human users, passwords may be extended to *passphrases*; in this case, the user types in a phrase or sentence rather than a short "word". The passphrase is hashed down to a fixed-size value, which plays the same role as a password; here, it is important that the passphrase is not simply truncated by the system, as passwords are in some systems. The idea is that users can remember phrases easier than random character sequences. If passwords resemble English text, then since each character contains only about 1.5 bits of entropy (Fact 7.67), a passphrase provides greater security through increased entropy than a short password. One drawback is the additional typing requirement.

10.2.2 Fixed password schemes: attacks

(i) Replay of fixed passwords

A weakness of schemes using fixed, reusable passwords (i.e., the basic scheme of §10.2), is the possibility that an adversary learns a user's password by observing it as it is typed in (or from where it may be written down). A second security concern is that user-entered passwords (or one-way hashes thereof) are transmitted in cleartext over the communications line between the user and the system, and are also available in cleartext temporarily during system verification. An eavesdropping adversary may record this data, allowing subsequent impersonation.

Fixed password schemes are thus of use when the password is transmitted over trusted communications lines safe from monitoring, but are not suitable in the case that passwords are transmitted over open communications networks. For example, in Figure 10.1, the claimant A may be a user logging in from home over a telephone modem, to a remote office site B two (or two thousand) miles away; the cleartext password might then travel over an unsecured telephone network (including possibly a wireless link), subject to eavesdropping.

In the case that remote identity verification is used for access to a local resource, e.g., an automated cash dispenser with on-line identity verification, the system response (accept/reject) must be protected in addition to the submitted password, and must include variability to prevent trivial replay of a time-invariant accept response.

(ii) Exhaustive password search

A very naive attack involves an adversary simply (randomly or systematically) trying passwords, one at a time, on the actual verifier, in hope that the correct password is found. This may be countered by ensuring passwords are chosen from a sufficiently large space, limiting the number of invalid (on-line) attempts allowed within fixed time periods, and slowing down the password mapping or login-process itself as in §10.2.1(iv). *Off-line attacks*, involving a (typically large) computation which does not require interacting with the actual verifier until a final stage, are of greater concern; these are now considered.

Given a password file containing one-way hashes of user passwords, an adversary may attempt to defeat the system by testing passwords one at a time, and comparing the one-way hash of each to passwords in the encrypted password file (see §10.2.1(ii)). This is theoretically possible since both the one-way mapping and the (guessed) plaintext are known. (This could be precluded by keeping any or all of the details of the one-way mapping or the password file itself secret, but it is not considered prudent to base the security of the system on the assumption that such details remain secret forever.) The feasibility of the attack depends on the number of passwords that need be checked before a match is expected (which itself depends on the number of possible passwords), and the time required to test each (see Example 10.4, Table 10.1, and Table 10.2). The latter depends on the password mapping used, its implementation, the instruction execution time of the host processor, and the number of processors available (note exhaustive search is parallelizable). The time required to actually compare the image of each trial password to all passwords in a password file is typically negligible.

10.4 Example (*password entropy*) Suppose passwords consist of strings of 7-bit ASCII characters. Each has a numeric value in the range 0-127. (When 8-bit characters are used, values 128-255 compose the *extended character set*, generally inaccessible from standard keyboards.) ASCII codes 0-31 are reserved for control characters; 32 is a space character; 33-126 are keyboard-accessible printable characters; and 127 is a special character. Table 10.1 gives the number of distinct n-character passwords composed of typical combinations of characters, indicating an upper bound on the security of such password spaces. □

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	23.5	25.9	29.8	32.9
6	28.2	31.0	35.7	39.4
7	32.9	36.2	41.7	46.0
8	37.6	41.4	47.6	52.6
9	42.3	46.5	53.6	59.1
10	47.0	51.7	59.5	65.7

Table 10.1: *Bitsize of password space for various character combinations. The number of n-character passwords, given c choices per character, is c^n. The table gives the base-2 logarithm of this number of possible passwords.*

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	0.67 hr	3.4 hr	51 hr	430 hr
6	17 hr	120 hr	130 dy	4.7 yr
7	19 dy	180 dy	22 yr	440 yr
8	1.3 yr	18 yr	1400 yr	42000 yr
9	34 yr	640 yr	86000 yr	4.0×10^6 yr
10	890 yr	23000 yr	5.3×10^6 yr	3.8×10^8 yr

Table 10.2: *Time required to search entire password space. The table gives the time T (in hours, days, or years) required to search or pre-compute over the entire specified spaces using a single processor (cf. Table 10.1). $T = c^n \cdot t \cdot y$, where t is the number of times the password mapping is iterated, and y the time per iteration, for $t = 25$, $y = 1/(125\ 000)$ sec. (This approximates the* UNIX *crypt command on a high-end PC performing DES at 1.0 Mbytes/s – see §10.2.3.)*

(iii) Password-guessing and dictionary attacks

To improve upon the expected probability of success of an exhaustive search, rather than searching through the space of all possible passwords, an adversary may search the space in order of decreasing (expected) probability. While ideally arbitrary strings of n characters would be equiprobable as user-selected passwords, most (unrestricted) users select passwords from a small subset of the full password space (e.g., short passwords; dictionary words; proper names; lowercase strings). Such weak passwords with low entropy are easily guessed; indeed, studies indicate that a large fraction of user-selected passwords are found in typical (intermediate) dictionaries of only 150 000 words, while even a large dictionary of 250 000 words represents only a tiny fraction of all possible n-character passwords (see Table 10.1).

Passwords found in any on-line or available list of words may be uncovered by an adversary who tries all words in this list, using a so-called *dictionary attack*. Aside from traditional dictionaries as noted above, on-line dictionaries of words from foreign languages, or on specialized topics such as music, film, etc. are available. For efficiency in repeated use by an adversary, an "encrypted" (hashed) list of dictionary or high-probability passwords may be created and stored on disk or tape; password images from system password files may then be collected, ordered (using a sorting algorithm or conventional hashing), and then compared to entries in the encrypted dictionary. Dictionary-style attacks are not generally successful at finding a particular user's password, but find many passwords in most systems.

10.2.3 Case study – UNIX passwords

The UNIX[2] operating system provides a widely known, historically important example of a fixed password system, implementing many of the ideas of §10.2.1. A UNIX password file contains a one-way function of user passwords computed as follows: each user password serves as the key to encrypt a known plaintext (64 zero-bits). This yields a one-way function of the key, since only the user (aside from the system, temporarily during password verification) knows the password. For the encryption algorithm, a minor modification of DES (§7.4) is used, as described below; variations may appear in products outside of the USA. The technique described relies on the conjectured property that DES is resistant to known-plaintext attacks – given cleartext and the corresponding ciphertext, it remains difficult to find the key.

The specific technique makes repeated use of DES, iterating the encipherment $t = 25$ times (see Figure 10.2). In detail, a user password is truncated to its first 8 ASCII characters. Each of these provides 7 bits for a 56-bit DES key (padded with 0-bits if less than 8 characters). The key is used to DES-encrypt the 64-bit constant 0, with the output fed back as input t times iteratively. The 64-bit result is repacked into 11 printable characters (a 64-bit output and 12 salt bits yields 76 bits; 11 ASCII characters allow 77). In addition, a non-standard method of password salting is used, intended to simultaneously complicate dictionary attacks and preclude use of off-the-shelf DES hardware for attacks:

1. *password salting*. UNIX password salting associates a 12-bit "random" salt (12 bits taken from the system clock at time of password creation) with each user-selected password. The 12 bits are used to alter the standard expansion function E of the DES mapping (see §7.4), providing one of 4096 variations. (The expansion E creates a 48-bit block; immediately thereafter, the salt bits collectively determine one of 4096 permutations. Each bit is associated with a pre-determined pair from the 48-bit block, e.g., bit 1 with block bits 1 and 25, bit 2 with block bits 2 and 26, etc. If the salt bit is 1, the block bits are swapped, and otherwise they are not.) Both the hashed password and salt are recorded in the system password file. Security of any particular user's password is unchanged by salting, but a dictionary attack now requires $2^{12} = 4096$ variations of each trial password.
2. *preventing use of off-the-shelf DES chips*. Because the DES expansion permutation E is dependent on the salt, standard DES chips can no longer be used to implement the UNIX password algorithm. An adversary wishing to use hardware to speed up an attack must build customized hardware rather than use commercially available chips. This may deter adversaries with modest resources.

The value stored for a given userid in the write-protected password file `/etc/passwd` is thus the iterated encryption of 0 under that user's password, using the salted modification of DES. The constant 0 here could be replaced by other values, but typically is not. The overall algorithm is called the UNIX *crypt* password algorithm.

[2] UNIX is a trademark of Bell Laboratories.

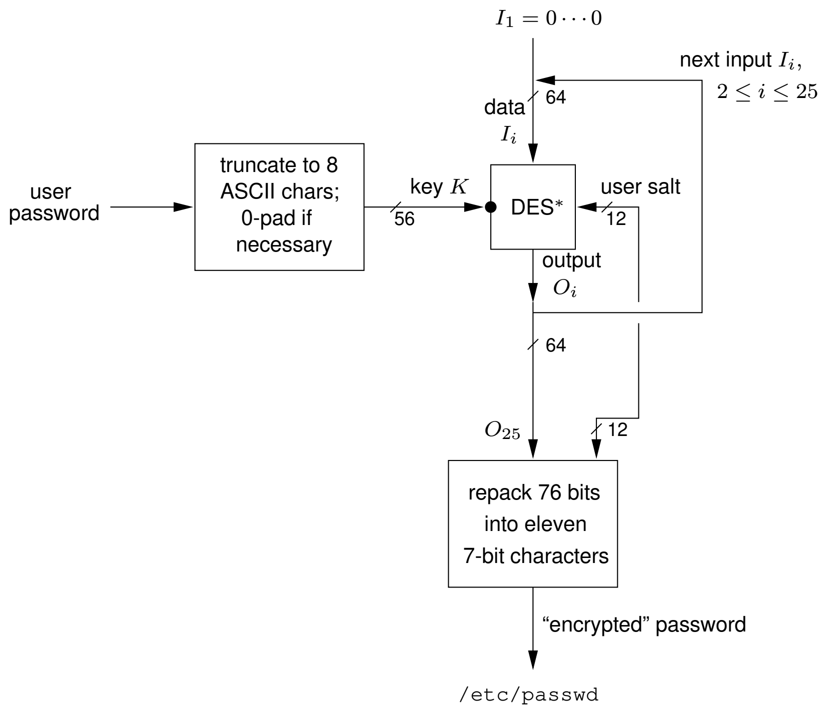

Figure 10.2: UNIX crypt *password mapping. DES* indicates DES with the expansion mapping E modified by a 12-bit salt.*

10.5 Remark (*performance advances*) While the UNIX *crypt* mapping with $t = 25$ iterations provided a reasonable measure of protection against exhaustive search when introduced in the 1970s, for equivalent security in a system designed today a more computationally intensive mapping would be provided, due to performance advances in both hardware and software.

10.2.4 PINs and passkeys

(i) PINs

Personal identification numbers (PINs) fall under the category of fixed (time-invariant) passwords. They are most often used in conjunction with "something possessed", typically a physical *token* such as a plastic banking card with a magnetic stripe, or a chipcard. To prove one's identity as the authorized user of the token, and gain access to the privileges associated therewith, entry of the correct PIN is required when the token is used. This provides a second level of security if the token is lost or stolen. PINs may also serve as the second level of security for entry to buildings which have an independent first level of security (e.g., a security guard or video camera).

For user convenience and historical reasons, PINs are typically short (relative to fixed password schemes) and numeric, e.g., 4 to 8 digits. To prevent exhaustive search through such a small key space (e.g., 10 000 values for a 4-digit numeric PIN), additional procedural constraints are necessary. For example, some automated cash dispenser machines accessed by banking cards confiscate a card if three incorrect PINs are entered successively; for others, incorrect entry of a number of successive PINs may cause the card to be "locked" or deactivated, thereafter requiring a longer PIN (e.g., 8 digits) for reactivation following such suspicious circumstances.

In an on-line system using PINs or reusable passwords, a claimed identity accompanied by a user-entered PIN may be verified by comparison to the PIN stored for that identity in a system database. An alternative is to use the PIN as a key for a MAC (see Chapter 9).

In an off-line system without access to a central database, information facilitating PIN verification must be stored on the token itself. If the PIN need not be user-selected, this may be done by defining the PIN to be a function of a secret key and the identity associated with the token; the PIN is then verifiable by any remote system knowing this master key.

In an off-line system, it may also be desirable to allow the PIN to be user-selectable, to facilitate PIN memorization by users. In this case, the PIN may be encrypted under a master key and stored on the token, with the master key known to all off-line terminals that need to be capable of verifying the token. A preferable design is to store a one-way function of the PIN, user identity, and master key on the token.

(ii) Two-stage authentication and password-derived keys

Human users have difficulty remembering secret keys which have sufficient entropy to provide adequate security. Two techniques which address this issue are now described.

When tokens are used with off-line PIN verification, a common technique is for the PIN to serve to verify the user to the token, while the token contains additional independent information allowing the token to authenticate itself to the system (as a valid token representing a legitimate user). The user is thereby indirectly authenticated to the system by a two-stage process. This requires the user have possession of the token but need remember only a short PIN, while a longer key (containing adequate entropy) provides cryptographic security for authentication over an unsecured link.

A second technique is for a user password to be mapped by a one-way hash function into a cryptographic key (e.g., a 56-bit DES key). Such password-derived keys are called *passkeys*. The passkey is then used to secure a communications link between the user and a system which also knows the user password. It should be ensured that the entropy of the user's password is sufficiently large that exhaustive search of the password space is not more efficient than exhaustive search of the passkey space (i.e., guessing passwords is not easier than guessing 56-bit DES keys); see Table 10.1 for guidance.

An alternative to having passkeys remain fixed until the password is changed is to keep a running sequence number on the system side along with each user's password, for use as a time-variant salt communicated to the user in the clear and incremented after each use. A fixed per-user salt could also be used in addition to a running sequence number.

Passkeys should be viewed as long-term keys, with use restricted to authentication and key management (e.g., rather than also for bulk encryption of user data). A disadvantage of using password-derived keys is that storing each user's password within the system requires some mechanism to protect the confidentiality of the stored passwords.

10.2.5 One-time passwords (towards strong authentication)

A natural progression from fixed password schemes to challenge-response identification protocols may be observed by considering one-time password schemes. As was noted in §10.2.2, a major security concern of fixed password schemes is eavesdropping and subsequent replay of the password. A partial solution is *one-time passwords*: each password is used only once. Such schemes are safe from passive adversaries who eavesdrop and later attempt impersonation. Variations include:

1. *shared lists of one-time passwords*. The user and the system use a sequence or set of t secret passwords, (each valid for a single authentication), distributed as a pre-shared list. A drawback is maintenance of the shared list. If the list is not used sequentially, the system may check the entered password against all remaining unused passwords. A variation involves use of a *challenge-response table*, whereby the user and the system share a table of matching challenge-response pairs, ideally with each pair valid at most once; this non-cryptographic technique differs from the cryptographic challenge-response of §10.3.
2. *sequentially updated one-time passwords*. Initially only a single secret password is shared. During authentication using password i, the user creates and transmits to the system a new password (password $i + 1$) encrypted under a key derived from password i. This method becomes difficult if communication failures occur.
3. *one-time password sequences based on a one-way function*. Lamport's one-time password scheme is described below. This method is more efficient (with respect to bandwidth) than sequentially updated one-time passwords, and may be viewed as a challenge-response protocol where the challenge is implicitly defined by the current position within the password sequence.

One-time passwords based on one-way functions (Lamport's scheme)

In Lamport's one-time password scheme, the user begins with a secret w. A one-way function (OWF) H is used to define the password sequence: $w, H(w), H(H(w)), \ldots, H^t(w)$. The password for the i^{th} identification session, $1 \le i \le t$, is defined to be $w_i = H^{t-i}(w)$.

10.6 Protocol Lamport's OWF-based one-time passwords

SUMMARY: A identifies itself to B using one-time passwords from a sequence.

1. *One-time setup*.
   (a) User A begins with a secret w. Let H be a one-way function.
   (b) A constant t is fixed (e.g., $t = 100$ or 1000), defining the number of identifications to be allowed. (The system is thereafter restarted with a new w, to avoid replay attacks.)
   (c) A transfers (the *initial shared secret*) $w_0 = H^t(w)$, in a manner guaranteeing its authenticity, to the system B. B initializes its counter for A to $i_A = 1$.
2. *Protocol messages*. The i^{th} identification, $1 \le i \le t$, proceeds as follows:

   $$A \rightarrow B: \quad A,\ i,\ w_i\ (= H^{t-i}(w)) \quad (1)$$

   Here $A \rightarrow B$: X denotes A sending the message X to B.
3. *Protocol actions*. To identify itself for session i, A does the following.
   (a) A's equipment computes $w_i = H^{t-i}(w)$ (easily done either from w itself, or from an appropriate intermediate value saved during the computation of $H^t(w)$ initially), and transmits (1) to B.
   (b) B checks that $i = i_A$, and that the received password w_i satisfies: $H(w_i) = w_{i-1}$. If both checks succeed, B accepts the password, sets $i_A \leftarrow i_A + 1$, and saves w_i for the next session verification.

10.7 Note (*pre-play attack*) Protocol 10.6 and similar one-time password schemes including that of Note 10.8 remain vulnerable to an active adversary who intercepts and traps (or impersonates the system in order to extract) an as-yet unused one-time password, for the purpose of subsequent impersonation. To prevent this, a password should be revealed only to a party which itself is known to be authentic. Challenge-response techniques (see §10.3) address this threat.

10.8 Note (*alternative one-time password scheme*) The following one-time-password alternative to Protocol 10.6 is suitable if storing actual passwords on the system side is acceptable (cf. Figure 10.1; compare also to §10.3.2(iii)). The claimant A has a shared password P with the system verifier B, to which it sends the data pair: $(r,\ H(r,P))$. The verifier computes the hash of the received value r and its local copy of P, and declares acceptance if this matches the received hash value. To avoid replay, r should be a sequence number, timestamp, or other parameter which can be easily guaranteed to be accepted only once.

10.3 Challenge-response identification (strong authentication)

The idea of cryptographic challenge-response protocols is that one entity (the claimant) "proves" its identity to another entity (the verifier) by demonstrating knowledge of a secret known to be associated with that entity, without revealing the secret itself to the verifier during the protocol.[3] This is done by providing a response to a time-variant challenge, where the response depends on both the entity's secret and the challenge. The *challenge* is typically a number chosen by one entity (randomly and secretly) at the outset of the protocol. If the communications line is monitored, the response from one execution of the identification protocol should not provide an adversary with useful information for a subsequent identification, as subsequent challenges will differ.

Before considering challenge-response identification protocols based on symmetric-key techniques (§10.3.2), public-key techniques (§10.3.3), and zero-knowledge concepts (§10.4), background on time-variant parameters is first provided.

10.3.1 Background on time-variant parameters

Time-variant parameters may be used in identification protocols to counteract replay and interleaving attacks (see §10.5), to provide uniqueness or timeliness guarantees, and to prevent certain chosen-text attacks. They may similarly be used in authenticated key establishment protocols (Chapter 12), and to provide uniqueness guarantees in conjunction with message authentication (Chapter 9).

Time-variant parameters which serve to distinguish one protocol instance from another are sometimes called *nonces*, *unique numbers*, or *non-repeating values*; definitions of these terms have traditionally been loose, as the specific properties required depend on the actual usage and protocol.

10.9 Definition A *nonce* is a value used no more than once for the same purpose. It typically serves to prevent (undetectable) replay.

[3] In some mechanisms, the secret is known to the verifier, and is used to verify the response; in others, the secret need not actually be known by the verifier.

The term *nonce* is most often used to refer to a "random" number in a challenge-response protocol, but the required randomness properties vary. Three main classes of time-variant parameters are discussed in turn below: random numbers, sequence numbers, and timestamps. Often, to ensure protocol security, the integrity of such parameters must be guaranteed (e.g., by cryptographically binding them with other data in a challenge-response sequence). This is particularly true of protocols in which the only requirement of a time-variant parameter is uniqueness, e.g., as provided by a never-repeated sequential counter.[4]

Following are some miscellaneous points about time-variant parameters.

1. Verifiable timeliness may be provided through use of random numbers in challenge-response mechanisms, timestamps in conjunction with distributed timeclocks, or sequence numbers in conjunction with the maintenance of pairwise (claimant, verifier) state information.
2. To provide timeliness or uniqueness guarantees, the verifier in the protocol controls the time-variant parameter, either directly (through choice of a random number) or indirectly (through information maintained regarding a shared sequence, or logically through a common time clock).
3. To uniquely identify a message or sequence of messages (protocol instance), nonces drawn from a monotonically increasing sequence may be used (e.g., sequence or serial numbers, and timestamps, if guaranteed to be increasing and unique), or random numbers of sufficient size. Uniqueness is often required only within a given key lifetime or time window.
4. Combinations of time-variant parameters may be used, e.g., random numbers concatenated to timestamps or sequence numbers. This may guarantee that a pseudorandom number is not duplicated.

(i) Random numbers

Random numbers may be used in challenge-response mechanisms, to provide uniqueness and timeliness assurances, and to preclude certain replay and interleaving attacks (see §10.5, including Remark 10.42). Random numbers may also serve to provide unpredictability, for example, to preclude chosen-text attacks.

The term *random numbers*, when used in the context of identification and authentication protocols, includes pseudorandom numbers which are unpredictable to an adversary (see Remark 10.11); this differs from randomness in the traditional statistical sense. In protocol descriptions, "choose a random number" is usually intended to mean "pick a number with uniform distribution from a specified sample space" or "select from a uniform distribution".

Random numbers are used in challenge-response protocols as follows. One entity includes a (new) random number in an outgoing message. An incoming message subsequently received (e.g., the next protocol message of the same protocol instance), whose construction required knowledge of this nonce and to which this nonce is inseparably bound, is then deemed to be *fresh* (Remark 10.10) based on the reasoning that the random number links the two messages. The non-tamperable binding is required to prevent appending a nonce to an old message.

Random numbers used in this manner serve to fix a relative point in time for the parties involved, analogous to a shared timeclock. The maximum allowable time between protocol messages is typically constrained by a *timeout period*, enforced using local, independent countdown timers.

[4]Such predictable parameters differ from sequence numbers in that they might not be bound to any stored state. Without appropriate cryptographic binding, a potential concern then is a pre-play attack wherein an adversary obtains the response before the time-variant parameter is legitimately sent (see Note 10.7).

10.10 Remark (*freshness*) In the context of challenge-response protocols, *fresh* typically means recent, in the sense of having originated subsequent to the beginning of the current protocol instance. Note that such freshness alone does not rule out interleaving attacks using parallel sessions (see §10.5).

10.11 Remark (*birthday repetitions in random numbers*) In generating pseudorandom numbers for use as time-variant parameters, it suffices if the probability of a repeated number is acceptably low and if numbers are not intentionally reused. This may be achieved by selecting the random value from a sufficiently large sample space, taking into account coincidences arising from the birthday paradox. The latter may be addressed by either using a larger sample space, or by using a generation process guaranteed to avoid repetition (e.g., a bijection), such as using the counter or OFB mode of a block cipher (§7.2.2).

10.12 Remark (*disadvantages of random numbers*) Many protocols involving random numbers require the generation of cryptographically secure (i.e., unpredictable) random numbers. If pseudorandom number generators are used, an initial seed with sufficient entropy is required. When random numbers are used in challenge-response mechanisms in place of timestamps, typically the protocol involves one additional message, and the challenger must temporarily maintain state information, but only until the response is verified.

(ii) Sequence numbers

A sequence number (serial number, or counter value) serves as a unique number identifying a message, and is typically used to detect message replay. For stored files, sequence numbers may serve as *version numbers* for the file in question. Sequence numbers are specific to a particular pair of entities, and must explicitly or implicitly be associated with both the originator and recipient of a message; distinct sequences are customarily necessary for messages from A to B and from B to A.

Parties follow a pre-defined policy for message numbering. A message is accepted only if the sequence number therein has not been used previously (or not used previously within a specified time period), and satisfies the agreed policy. The simplest policy is that a sequence number starts at zero, is incremented sequentially, and each successive message has a number one greater than the previous one received. A less restrictive policy is that sequence numbers need (only) be monotonically increasing; this allows for lost messages due to non-malicious communications errors, but precludes detection of messages lost due to adversarial intervention.

10.13 Remark (*disadvantages of sequence numbers*) Use of sequence numbers requires an overhead as follows: each claimant must record and maintain long-term pairwise state information for each possible verifier, sufficient to determine previously used and/or still valid sequence numbers. Special procedures (e.g., for resetting sequence numbers) may be necessary following circumstances disrupting normal sequencing (e.g., system failures). Forced delays are not detectable in general. As a consequence of the overhead and synchronization necessary, sequence numbers are most appropriate for smaller, closed groups.

(iii) Timestamps

Timestamps may be used to provide timeliness and uniqueness guarantees, to detect message replay. They may also be used to implement time-limited access privileges, and to detect forced delays.

Timestamps function as follows. The party originating a message obtains a timestamp from its local (host) clock, and cryptographically binds it to a message. Upon receiving a time-stamped message, the second party obtains the current time from its own (host) clock, and subtracts the timestamp received. The received message is valid provided:

1. the timestamp difference is within the *acceptance window* (a fixed-size time interval, e.g., 10 milliseconds or 20 seconds, selected to account for the maximum message transit and processing time, plus clock skew); and
2. (optionally) no message with an identical timestamp has been previously received from the same originator. This check may be made by the verifier maintaining a list of all timestamps received from each source entity within the current acceptance window. Another method is to record the latest (valid) timestamp used by each source (in this case the verifier accepts only strictly increasing time values).

The security of timestamp-based verification relies on use of a common time reference. This requires that host clocks be available and both "loosely synchronized" and secured from modification. Synchronization is necessary to counter clock drift, and must be appropriate to accommodate the acceptance window used. The degree of clock skew allowed, and the acceptance window, must be appropriately small to preclude message replay if the above optional check is omitted. The timeclock must be secure to prevent adversarial resetting of a clock backwards so as to restore the validity of old messages, or setting a clock forward to prepare a message for some future point in time (cf. Note 10.7).

10.14 Remark (*disadvantages of timestamps*) Timestamp-based protocols require that timeclocks be both synchronized and secured. The preclusion of adversarial modification of local timeclocks is difficult to guarantee in many distributed environments; in this case, the security provided must be carefully re-evaluated. Maintaining lists of used timestamps within the current window has the drawback of a potentially large storage requirement, and corresponding verification overhead. While technical solutions exist for synchronizing distributed clocks, if synchronization is accomplished via network protocols, such protocols themselves must be secure, which typically requires authentication; this leads to a circular security argument if such authentication is itself timestamp-based.

10.15 Remark (*comparison of time-variant parameters*) Timestamps in protocols offer the advantage of fewer messages (typically by one), and no requirement to maintain pairwise long-term state information (cf. sequence numbers) or per-connection short-term state information (cf. random numbers). Minimizing state information is particularly important for servers in client-server applications. The main drawback of timestamps is the requirement of maintaining secure, synchronized distributed timeclocks. Timestamps in protocols may typically be replaced by a random number challenge plus a return message.

10.3.2 Challenge-response by symmetric-key techniques

Challenge-response mechanisms based on symmetric-key techniques require the claimant and the verifier to share a symmetric key. For closed systems with a small number of users, each pair of users may share a key a priori; in larger systems employing symmetric-key techniques, identification protocols often involve the use of a trusted on-line server with which each party shares a key. The on-line server effectively acts like the hub of a spoked wheel, providing a common session key to two parties each time one requests authentication with the other.

The apparent simplicity of the techniques presented below and in §10.3.3 is misleading. The design of such techniques is intricate and the security is brittle; those presented have been carefully selected.

(i) Challenge-response based on symmetric-key encryption

Both the Kerberos protocol (Protocol 12.24) and the Needham-Schroeder shared-key protocol (Protocol 12.26) provide entity authentication based on symmetric encryption and involve use of an on-line trusted third party. These are discussed in Chapter 12, as they additionally provide key establishment.

Below, three simple techniques based on ISO/IEC 9798-2 are described. They assume the prior existence of a shared secret key (and no further requirement for an on-line server). In this case, two parties may carry out unilateral entity authentication in one pass using timestamps or sequence numbers, or two passes using random numbers; mutual authentication requires, respectively, two and three passes. The claimant corroborates its identity by demonstrating knowledge of the shared key by encrypting a challenge (and possibly additional data) using the key. These techniques are similar to those given in §12.3.1.

10.16 Remark (*data integrity*) When encipherment is used in entity authentication protocols, data integrity must typically also be guaranteed to ensure security. For example, for messages spanning more than one block, the rearrangement of ciphertext blocks cannot be detected in the ECB mode of block encryption, and even CBC encryption may provide only a partial solution. Such data integrity should be provided through use of an accepted data integrity mechanism (see §9.6; cf. Remark 12.19).

9798-2 mechanisms: Regarding notation: r_A and t_A, respectively, denote a random number and a timestamp, generated by A. (In these mechanisms, the timestamp t_A may be replaced by a sequence number n_A, providing slightly different guarantees.) E_K denotes a symmetric encryption algorithm, with a key K shared by A and B; alternatively, distinct keys K_{AB} and K_{BA} may be used for unidirectional communication. It is assumed that both parties are aware of the claimed identity of the other, either by context or by additional (unsecured) cleartext data fields. Optional message fields are denoted by an asterisk (*), while a comma (,) within the scope of E_K denotes concatenation.

1. *unilateral authentication, timestamp-based*:

$$A \to B : E_K(t_A, B^*) \quad (1)$$

   Upon reception and decryption, B verifies that the timestamp is acceptable, and optionally verifies the received identifier as its own. The identifier B here prevents an adversary from re-using the message immediately on A, in the case that a single bi-directional key K is used.

2. *unilateral authentication, using random numbers*:
   To avoid reliance on timestamps, the timestamp may be replaced by a random number, at the cost of an additional message:

$$\begin{array}{ll} A \leftarrow B : r_B & (1) \\ A \to B : E_K(r_B, B^*) & (2) \end{array}$$

   B decrypts the received message and checks that the random number matches that sent in (1). Optionally, B checks that the identifier in (2) is its own; this prevents a reflection attack in the case of a bi-directional key K. To prevent chosen-text attacks on the encryption scheme E_K, A may (as below) embed an additional random number in (2) or, alternately, the form of the challenges can be restricted; the critical requirement is that they be non-repeating.

3. *mutual authentication, using random numbers*:

$$\begin{array}{ll} A \leftarrow B : r_B & (1) \\ A \rightarrow B : E_K(r_A, r_B, B^*) & (2) \\ A \leftarrow B : E_K(r_B, r_A) & (3) \end{array}$$

Upon reception of (2), B carries out the checks as above and, in addition, recovers the decrypted r_A for inclusion in (3). Upon decrypting (3), A checks that both random numbers match those used earlier. The second random number r_A in (2) serves both as a challenge and to prevent chosen-text attacks.

10.17 Remark (*doubling unilateral authentication*) While mutual authentication may be obtained by running any of the above unilateral authentication mechanisms twice (once in each direction), such an ad-hoc combination suffers the drawback that the two unilateral authentications, not being linked, cannot logically be associated with a single protocol run.

(ii) Challenge-response based on (keyed) one-way functions

The encryption algorithm in the above mechanisms may be replaced by a one-way or non-reversible function of the shared key and challenge, e.g., having properties similar to a MAC (Definition 9.7). This may be preferable in situations where encryption algorithms are otherwise unavailable or undesirable (e.g., due to export restrictions or computational costs). The modifications required to the 9798-2 mechanisms above (yielding the analogous mechanisms of ISO/IEC 9798-4) are the following:

1. the encryption function E_K is replaced by a MAC algorithm h_K;
2. rather than decrypting and verifying that fields match, the recipient now independently computes the MAC value from known quantities, and accepts if the computed MAC matches the received MAC value; and
3. to enable independent MAC computation by the recipient, the additional cleartext field t_A must be sent in message (1) of the one-pass mechanism. r_A must be sent as an additional cleartext field in message (2) of the three-pass mechanism.

The revised three-pass challenge-response mechanism based on a MAC h_K, with actions as noted above, provides mutual identification. Essentially the same protocol, called *SKID3*, has messages as follows:

$$\begin{array}{lll} A \leftarrow B : & r_B & (1) \\ A \rightarrow B : & r_A,\ h_K(r_A, r_B, B) & (2) \\ A \leftarrow B : & h_K(r_B, r_A, A) & (3) \end{array}$$

Note that the additional field A is included in message (3). The protocol *SKID2*, obtained by omitting the third message, provides unilateral entity authentication.

(iii) Implementation using hand-held passcode generators

Answering a challenge in challenge-response protocols requires some type of computing device and secure storage for long-term keying material (e.g., a file on a trusted local disk, perhaps secured under a local password-derived key). For additional security, a device such as a chipcard (and corresponding card reader) may be used for both the key storage and response computation. In some cases, a less expensive option is a passcode generator.

Passcode generators are hand-held devices, resembling thin calculators in both size and display, and which provide time-variant passwords or *passcodes* (see Figure 10.3). The generator contains a device-specific secret key. When a user is presented with a challenge (e.g., by a system displaying it on a computer terminal), the challenge is keyed into the generator. The generator displays a passcode, computed as a function of the secret key and the challenge; this may be either an asymmetric function, or a symmetric function (e.g., encryption or MAC as discussed above). The user returns the response (e.g., keys the passcode in at his terminal), which the system verifies by comparison to an independently computed response, using the same information stored on the system side.

For further protection against misplaced generators, the response may also depend on a user-entered PIN. Simpler passcode generators omit the user keypad, and use as an implicit challenge a time value (with a typical granularity of one minute) defined by a timeclock loosely synchronized automatically between the system and the passcode generator. A more sophisticated device combines implicit synchronization with explicit challenges, presenting an explicit challenge only when synchronization is lost.

A drawback of systems using passcode generators is, as per §10.2.1(i), the requirement to provide confidentiality for user passwords stored on the system side.

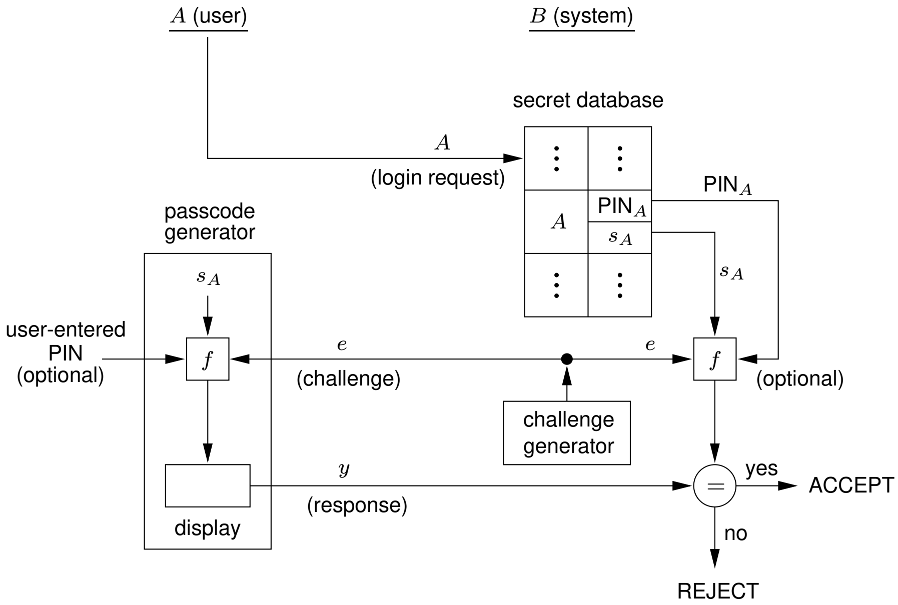

Figure 10.3: *Functional diagram of a hand-held passcode generator. s_A is A's user-specific secret. f is a one-way function. The (optional) PIN could alternatively be locally verified in the passcode generator only, making y independent of it.*

10.3.3 Challenge-response by public-key techniques

Public-key techniques may be used for challenge-response based identification, with a claimant demonstrating knowledge of its private key in one of two ways (cf. §12.5):

1. the claimant decrypts a challenge encrypted under its public key;
2. the claimant digitally signs a challenge.

Ideally, the public-key pair used in such mechanisms should not be used for other purposes, since combined usage may compromise security (Remark 10.40). A second caution is that the public-key system used should not be susceptible to chosen-ciphertext attacks,[5]

[5]Both chosen-ciphertext and chosen-plaintext attacks are of concern for challenge-response techniques based on symmetric-key encryption.

as an adversary may attempt to extract information by impersonating a verifier and choosing strategic rather than random challenges. (See Notes 8.13 and 8.58 regarding the Rabin/Williams and Blum-Goldwasser schemes.)

Incorporating a self-generated random number or *confounder* (§10.5) into the data over which the response is computed may address both of these concerns. Such data may be made available to the verifier in cleartext to allow verification.

(i) Challenge-response based on public-key decryption

Identification based on PK decryption and witness. Consider the following protocol:

$$\begin{array}{lll} A \leftarrow B: & h(r), B, P_A(r, B) & (1) \\ A \rightarrow B: & r & (2) \end{array}$$

B chooses a random r, computes the *witness* $x = h(r)$ (x demonstrates knowledge of r without disclosing it – cf. §10.4.1), and computes the challenge $e = P_A(r, B)$. Here P_A denotes the public-key encryption (e.g., RSA) algorithm of A, and h denotes a one-way hash function. B sends (1) to A. A decrypts e to recover r' and B', computes $x' = h(r')$, and quits if $x' \neq x$ (implying $r' \neq r$) or if B' is not equal to its own identifier B. Otherwise, A sends $r = r'$ to B. B succeeds with (unilateral) entity authentication of A upon verifying the received r agrees with that sent earlier. The use of the witness precludes chosen-text attacks.

Modified Needham-Schroeder PK protocol for identification. The modified Needham-Schroeder public-key protocol of Note 12.39 provides key transport of distinct keys k_1, k_2 from A to B and B to A, respectively, as well as mutual authentication. If the key establishment feature is not required, k_1 and k_2 may be omitted. With P_B denoting the public-key encryption algorithm for B (e.g., RSA), the messages in the modified protocol for identification are then as follows:

$$\begin{array}{lll} A \rightarrow B: & P_B(r_1, A) & (1) \\ A \leftarrow B: & P_A(r_1, r_2) & (2) \\ A \rightarrow B: & r_2 & (3) \end{array}$$

Verification actions are analogous to those of Note 12.39.

(ii) Challenge-response based on digital signatures

X.509 mechanisms based on digital signatures. The ITU-T (formerly CCITT) X.509 two- and three-way strong authentication protocols specify identification techniques based on digital signatures and, respectively, timestamps and random number challenges. These are described in §12.5.2, and optionally provide key establishment in addition to entity authentication.

9798-3 mechanisms. Three challenge-response identification mechanisms based on signatures are given below, analogous to those in §10.3.2(i) based on symmetric-key encryption, but, in this case, corresponding to techniques in ISO/IEC 9798-3. Regarding notation (cf. 9798-2 above): r_A and t_A, respectively, denote a random number and timestamp generated by A. S_A denotes A's signature mechanism; if this mechanism provides message recovery, some of the cleartext fields listed below are redundant and may be omitted. $cert_A$ denotes the public-key certificate containing A's signature public key. (In these mechanisms, if the verifier has the authentic public key of the claimant a priori, certificates may be omitted; otherwise, it is assumed that the verifier has appropriate information to verify the validity of the public key contained in a received certificate – see Chapter 13.) Remark 10.17 also applies here.

1. *unilateral authentication with timestamps*:

$$A \to B : cert_A, t_A, B, S_A(t_A, B) \quad (1)$$

Upon reception, B verifies that the timestamp is acceptable, the received identifier B is its own, and (using A's public key extracted from $cert_A$ after verifying the latter) checks that the signature over these two fields is correct.

2. *unilateral authentication with random numbers*: Reliance on timestamps may be replaced by a random number, at the cost of an additional message:

$$\begin{array}{ll} A \leftarrow B : r_B & (1) \\ A \to B : cert_A, r_A, B, S_A(r_A, r_B, B) & (2) \end{array}$$

B verifies that the cleartext identifier is its own, and using a valid signature public key for A (e.g., from $cert_A$), verifies that A's signature is valid over the cleartext random number r_A, the same number r_B as sent in (1), and this identifier. The signed r_A explicitly prevents chosen-text attacks.

3. *mutual authentication with random numbers*:

$$\begin{array}{ll} A \leftarrow B : r_B & (1) \\ A \to B : cert_A, r_A, B, S_A(r_A, r_B, B) & (2) \\ A \leftarrow B : cert_B, A, S_B(r_B, r_A, A) & (3) \end{array}$$

Processing of (1) and (2) is as above; (3) is processed analogously to (2).

10.4 Customized and zero-knowledge identification protocols

This section considers protocols specifically designed to achieve identification, which use asymmetric techniques but do not rely on digital signatures or public-key encryption, and which avoid use of block ciphers, sequence numbers, and timestamps. They are similar in some regards to the challenge-response protocols of §10.3, but are based on the ideas of interactive proof systems and zero-knowledge proofs (see §10.4.1), employing random numbers not only as challenges, but also as *commitments* to prevent cheating.

10.4.1 Overview of zero-knowledge concepts

A disadvantage of simple password protocols is that when a claimant A (called a *prover* in the context of zero-knowledge protocols) gives the verifier B her password, B can thereafter impersonate A. Challenge-response protocols improve on this: A responds to B's challenge to demonstrate knowledge of A's secret in a time-variant manner, providing information not directly reusable by B. This might nonetheless reveal some partial information about the claimant's secret; an adversarial verifier might also be able to strategically select challenges to obtain responses providing such information (see chosen-text attacks, §10.5).

Zero-knowledge (ZK) protocols are designed to address these concerns, by allowing a prover to demonstrate knowledge of a secret while revealing no information whatsoever (beyond what the verifier was able to deduce prior to the protocol run) of use to the verifier in conveying this demonstration of knowledge to others. The point is that only a single bit of information need be conveyed – namely, that the prover actually does know the secret.

More generally, a zero-knowledge protocol allows a proof of the truth of an assertion, while conveying no information whatsoever (this notion can be quantified in a rigorous sense) about the assertion itself other than its actual truth. In this sense, a zero-knowledge proof is similar to an answer obtained from a (trusted) *oracle*.

(i) Interactive proof systems and zero-knowledge protocols

The ZK protocols to be discussed are instances of *interactive proof systems*, wherein a prover and verifier exchange multiple messages (challenges and responses), typically dependent on random numbers (ideally: the outcomes of fair coin tosses) which they may keep secret. The prover's objective is to convince (*prove* to) the verifier the truth of an assertion, e.g., claimed knowledge of a secret. The verifier either accepts or rejects the *proof*. The traditional mathematical notion of a proof, however, is altered to an interactive game wherein proofs are *probabilistic* rather than absolute; a proof in this context need be correct only with bounded probability, albeit possibly arbitrarily close to 1. For this reason, an interactive proof is sometimes called a *proof by protocol*.

Interactive proofs used for identification may be formulated as proofs of knowledge. A possesses some secret s, and attempts to convince B it has *knowledge* of s by correctly responding to queries (involving publicly known inputs and agreed upon functions) which require knowledge of s to answer. Note that proving knowledge of s differs from proving that such s exists – for example, proving knowledge of the prime factors of n differs from proving that n is composite.

An interactive proof is said to be a *proof of knowledge* if it has both the properties of completeness and soundness. Completeness may be viewed as the customary requirement that a protocol functions properly given honest participants.

10.18 Definition (*completeness property*) An interactive proof (protocol) is *complete* if, given an honest prover and an honest verifier, the protocol succeeds with overwhelming probability (i.e., the verifier accepts the prover's claim). The definition of *overwhelming* depends on the application, but generally implies that the probability of failure is not of practical significance.

10.19 Definition (*soundness property*) An interactive proof (protocol) is *sound* if there exists an expected polynomial-time algorithm M with the following property: if a dishonest prover (impersonating A) can with non-negligible probability successfully execute the protocol with B, then M can be used to extract from this prover knowledge (essentially equivalent to A's secret) which with overwhelming probability allows successful subsequent protocol executions.

An alternate explanation of the condition in Definition 10.19 is as follows: the prover's secret s together with public data satisfies some polynomial-time predicate, and another solution of this predicate (possibly the same) can be extracted, allowing successful execution of subsequent protocol instances.

Since any party capable of impersonating A must know the equivalent of A's secret knowledge (M can be used to extract it from this party in polynomial time), soundness guarantees that the protocol does indeed provide a proof of knowledge – knowledge equivalent to that being queried is required to succeed. Soundness thus prevents a dishonest prover from convincing an honest verifier (but does does not by itself guarantee that acquiring the prover's secret is difficult; see Remark 10.23). A standard method to establish the soundness of a particular protocol is to assume the existence of a dishonest prover capable of successfully executing the protocol, and show how this allows one to compute the real prover's secret.

While an interactive proof of knowledge (or protocol based thereon) must be sound to be of cryptographic use, the main property of zero-knowledge protocols is the zero-knowledge aspect itself. For what follows, define a *transcript* (or view) to be the collection of messages resulting from protocol execution.

10.20 Definition (*zero-knowledge property*) A protocol which is a proof of knowledge has the *zero-knowledge property* if it is simulatable in the following sense: there exists an expected polynomial-time algorithm (*simulator*) which can produce, upon input of the assertion(s) to be proven but without interacting with the real prover, transcripts indistinguishable from those resulting from interaction with the real prover.

The zero-knowledge property implies that a prover executing the protocol (even when interacting with a malicious verifier) does not release any information (about its secret knowledge, other than that the particular assertion itself is true) not otherwise computable in polynomial time from public information alone. Thus, participation does not increase the chances of subsequent impersonation.

10.21 Remark (*simulated ZK protocols and protocol observers*) Consider an observer C who witnesses a zero-knowledge interactive proof (ZKIP) involving a prover A convincing a verifier B ($B \neq C$) of some knowledge A has. The "proof" to B does not provide any guarantees to C. (Indeed, A and B might have a prior agreement, conspiring against C, on the challenges to be issued.) Similarly, a recorded ZKIP conveys no guarantees upon playback. This is fundamental to the idea of the zero-knowledge property and the condition that proofs be simulatable by a verifier alone. Interactive proofs convey knowledge only to (interactive) verifiers able to select their own random challenges.

10.22 Definition (*computational vs. perfect zero-knowledge*) A protocol is *computationally* zero-knowledge if an observer restricted to probabilistic polynomial-time tests cannot distinguish real from simulated transcripts. For *perfect* zero-knowledge, the probability distributions of the transcripts must be identical. By convention, when not further qualified, *zero-knowledge* means computational zero-knowledge.

In the case of computational zero-knowledge, real and simulated transcripts are said to be *polynomially indistinguishable* (indistinguishable using polynomial-time algorithms). Any information extracted by a verifier through interaction with a prover provides no advantage to the verifier within polynomial time.

10.23 Remark (*ZK property and soundness vs. security*) The zero-knowledge property (Definition 10.20) does not guarantee that a protocol is secure (i.e., that the probability of it being easily defeated is negligible). Similarly, the soundness property (Definition 10.19) does not guarantee that a protocol is secure. Neither property has much value unless the underlying problem faced by an adversary is computationally hard.

(ii) Comments on zero-knowledge vs. other asymmetric protocols

The following observations may be made regarding zero-knowledge (ZK) techniques, as compared with other public-key (PK) techniques.

1. *no degradation with usage*: protocols proven to have the ZK property do not suffer degradation of security with repeated use, and resist chosen-text attacks. This is perhaps the most appealing practical feature of ZK techniques.
   A ZK technique which is not provably secure may or may not be viewed as more desirable than a PK technique which is provably secure (e.g., as difficult as factoring).
2. *encryption avoided*: many ZK techniques avoid use of explicit encryption algorithms. This may offer political advantages (e.g., with respect to export controls).
3. *efficiency*: while some ZK-based techniques are extremely efficient (see §10.4.5), protocols which formally have the zero-knowledge property typically have higher communications and/or computational overheads than PK protocols which do not. The computational efficiency of the more practical ZK-based schemes arises from their nature as interactive proofs, rather than their zero-knowledge aspect.
4. *unproven assumptions*: many ZK protocols ("proofs of knowledge") themselves rely on the same unproven assumptions as PK techniques (e.g., the intractability of factoring or quadratic residuosity).
5. *ZK-based vs. ZK*: although supported by prudent underlying principles, many techniques based on zero-knowledge concepts fall short of formally being zero-knowledge and/or formally sound in practice, due to parameter selection for reasons of efficiency, or for other technical reasons (cf. Notes 10.33 and 10.38). In fact, many such concepts are asymptotic, and do not apply directly to practical protocols (Remark 10.34).

(iii) Example of zero-knowledge proof: Fiat-Shamir identification protocol

The general idea of a zero-knowledge (ZK) proof is illustrated by the basic version of the Fiat-Shamir protocol. The basic version is presented here for historical and illustrative purposes (Protocol 10.24). In practice, one would use a more efficient variation, such as Protocol 10.26, with multiple "questions" per iteration rather than as here, where B poses only a single one-bit challenge per iteration.

The objective is for A to identify itself by proving knowledge of a secret s (associated with A through authentic public data) to any verifier B, without revealing any information about s not known or computable by B prior to execution of the protocol (see Note 10.25). The security relies on the difficulty of extracting square roots modulo large composite integers n of unknown factorization, which is equivalent to that of factoring n (Fact 3.46).

10.24 Protocol Fiat-Shamir identification protocol (basic version)

SUMMARY: A proves knowledge of s to B in t executions of a 3-pass protocol.

1. *One-time setup.*
   (a) A trusted center T selects and publishes an RSA-like modulus $n = pq$ but keeps primes p and q secret.
   (b) Each claimant A selects a secret s coprime to n, $1 \le s \le n-1$, computes $v = s^2 \bmod n$, and registers v with T as its public key.[6]
2. *Protocol messages.* Each of t rounds has three messages with form as follows.

$$\begin{array}{lll} A \rightarrow B: & x = r^2 \bmod n & (1) \\ A \leftarrow B: & e \in \{0,1\} & (2) \\ A \rightarrow B: & y = r \cdot s^e \bmod n & (3) \end{array}$$

[6]Technically, T should verify the condition $\gcd(s, n) = 1$ or equivalently $\gcd(v, n) = 1$, for this to be a sound proof of knowledge; and B should stop with failure if $\gcd(y, n) \neq 1$, where y is A's response in the third message. But either condition failing would allow the factorization of n, violating the assumption that n cannot be factored.

3. *Protocol actions*. The following steps are iterated t times (sequentially and independently). B accepts the proof if all t rounds succeed.
   (a) A chooses a random (*commitment*) r, $1 \le r \le n-1$, and sends (the *witness*) $x = r^2 \bmod n$ to B.
   (b) B randomly selects a (*challenge*) bit $e = 0$ or $e = 1$, and sends e to A.
   (c) A computes and sends to B (the *response*) y, either $y = r$ (if $e = 0$) or $y = rs \bmod n$ (if $e = 1$).
   (d) B rejects the proof if $y = 0$, and otherwise accepts upon verifying $y^2 \equiv x \cdot v^e \pmod{n}$. (Depending on e, $y^2 = x$ or $y^2 = xv \bmod n$, since $v = s^2 \bmod n$. Note that checking for $y = 0$ precludes the case $r = 0$.)

Protocol 10.24 may be explained and informally justified as follows. The challenge (or *exam*) e requires that A be capable of answering two questions, one of which demonstrates her knowledge of the secret s, and the other an easy question (for honest provers) to prevent cheating. An adversary impersonating A might try to cheat by selecting any r and setting $x = r^2/v$, then answering the challenge $e = 1$ with a "correct" answer $y = r$; but would be unable to answer the exam $e = 0$ which requires knowing a square root of $x \bmod n$. A prover A knowing s can answer both questions, but otherwise can at best answer one of the two questions, and so has probability only $1/2$ of escaping detection. To decrease the probability of cheating arbitrarily to an acceptably small value of 2^{-t} (e.g., $t = 20$ or $t = 40$), the protocol is iterated t times, with B accepting A's identity only if all t questions (over t rounds) are successfully answered.

10.25 Note (*secret information revealed by A*) The response $y = r$ is independent of A's secret s, while the response $y = rs \bmod n$ also provides no information about s because the random r is unknown to B. Information pairs (x, y) extracted from A could equally well be simulated by a verifier B alone by choosing y randomly, then defining $x = y^2$ or $y^2/v \bmod n$. While this is not the method by which A would construct such pairs, such pairs (x, y) have a probability distribution which is indistinguishable from those A would produce; this establishes the zero-knowledge property. Despite the ability to simulate proofs, B is unable to impersonate A because B cannot predict the real-time challenges.

As a minor technical point, however, the protocol does reveal a bit of information: the answer $y = rs$ provides supporting evidence that v is indeed a square modulo n, and the soundness of the protocol allows one to conclude, after t successful iterations, that this is indeed the case.

(iv) General structure of zero-knowledge protocols

Protocol 10.24 illustrates the general structure of a large class of three-move zero-knowledge protocols:

$$\begin{aligned} A \rightarrow B &: \quad witness \\ A \leftarrow B &: \quad challenge \\ A \rightarrow B &: \quad response \end{aligned}$$

The prover claiming to be A selects a random element from a pre-defined set as its secret commitment (providing hidden randomization or "private coin tosses"), and from this computes an associated (public) witness. This provides initial randomness for variation from other protocol runs, and essentially defines a set of questions all of which the prover claims to be able to answer, thereby a priori constraining her forthcoming response. By protocol design, only the legitimate party A, with knowledge of A's secret, is truly capable of answering all the questions, and the answer to any one of these provides no information about

A's long-term secret. B's subsequent challenge selects one of these questions. A provides its response, which B checks for correctness. The protocol is iterated, if necessary, to improve the bound limiting the probability of successful cheating.

Zero-knowledge interactive protocols thus combine the ideas of *cut-and-choose* protocols (this terminology results from the standard method by which two children share a piece of cake: one cuts, the other chooses) and challenge-response protocols. A responds to at most one challenge (question) for a given witness, and should not reuse any witness; in many protocols, security (possibly of long-term keying material) may be compromised if either of these conditions is violated.

10.4.2 Feige-Fiat-Shamir identification protocol

The basic version of the Fiat-Shamir protocol is presented as Protocol 10.24. This can be generalized, and the Feige-Fiat-Shamir (FSS) identification protocol (Protocol 10.26) is a minor variation of such a generalization. The FFS protocol involves an entity identifying itself by proving knowledge of a secret using a zero-knowledge proof; the protocol reveals no partial information whatsoever regarding the secret identification value(s) of A (cf. Definition 10.20). It requires limited computation (a small fraction of that required by RSA – see §10.4.5), and is thus well-suited for applications with low-power processors (e.g., 8-bit chipcard microprocessors).

10.26 Protocol Feige-Fiat-Shamir identification protocol

SUMMARY: A proves its identity to B in t executions of a 3-pass protocol.

1. *Selection of system parameters.* A trusted center T publishes the common modulus $n = pq$ for all users, after selecting two secret primes p and q each congruent to $3 \bmod 4$, and such that n is computationally infeasible to factor. (Consequently, n is a Blum integer per §2.4.6, and -1 is a quadratic non-residue mod n with Jacobi symbol $+1$.) Integers k and t are defined as security parameters (see Note 10.28).
2. *Selection of per-entity secrets.* Each entity A does the following.
   (a) Select k random integers $s_1, s_2, \dots, s_k$ in the range $1 \le s_i \le n-1$, and k random bits $b_1, \dots, b_k$. (For technical reasons, $\gcd(s_i, n) = 1$ is required, but is almost surely guaranteed as its failure allows factorization of n.)
   (b) Compute $v_i = (-1)^{b_i} \cdot (s_i^2)^{-1} \bmod n$ for $1 \le i \le k$. (This allows v_i to range over all integers coprime to n with Jacobi symbol $+1$, a technical condition required to prove that no secret information is "leaked"; by choice of n, precisely one signed choice for v_i has a square root.)
   (c) A identifies itself by non-cryptographic means (e.g., photo id) to T, which thereafter registers A's public key $(v_1, \dots, v_k; n)$, while only A knows its private key $(s_1, \dots, s_k)$ and n. (To guarantee the bounded probability of attack specified per Note 10.28, T may confirm that each v_i indeed does have Jacobi symbol +1 relative to n.) This completes the one-time set-up phase.
3. *Protocol messages.* Each of t rounds has three messages with form as follows.

$$\begin{array}{lll} A \to B: & x \ (= \pm r^2 \bmod n) & (1) \\ A \leftarrow B: & (e_1, \dots, e_k),\ e_i \in \{0,1\} & (2) \\ A \to B: & y \ (= r \cdot \prod_{e_j=1} s_j \bmod n) & (3) \end{array}$$

4. *Protocol actions.* The following steps are executed t times; B accepts A's identity if all t rounds succeed. Assume B has A's authentic public key $(v_1, \dots, v_k; n)$; otherwise, a certificate may be sent in message (1), and used as in Protocol 10.36.

(a) A chooses a random integer r, $1 \le r \le n-1$, and a random bit b; computes $x = (-1)^b \cdot r^2 \bmod n$; and sends x (the *witness*) to B.
(b) B sends to A (the *challenge*,) a random k-bit vector $(e_1, \dots, e_k)$.
(c) A computes and sends to B (the *response*): $y = r \cdot \prod_{j=1}^{k} s_j^{e_j} \bmod n$ (the product of r and those s_j specified by the challenge).
(d) B computes $z = y^2 \cdot \prod_{j=1}^{k} v_j^{e_j} \bmod n$, and verifies that $z = \pm x$ and $z \ne 0$. (The latter precludes an adversary succeeding by choosing $r = 0$.)

10.27 Example (*Feige-Fiat-Shamir protocol with artificially small parameters*)

1. The trusted center T selects the primes $p = 683$, $q = 811$, and publishes $n = pq = 553913$. Integers $k = 3$ and $t = 1$ are defined as security parameters.
2. Entity A does the following.
   (a) Selects 3 random integers $s_1 = 157$, $s_2 = 43215$, $s_3 = 4646$, and 3 bits $b_1 = 1$, $b_2 = 0$, $b_3 = 1$.
   (b) Computes $v_1 = 441845$, $v_2 = 338402$, and $v_3 = 124423$.
   (c) A's public key is $(441845, 338402, 124423; 553913)$ and private key is $(157, 43215, 4646)$.
3. See Protocol 10.26 for a summary of the messages exchanged.
4. (a) A chooses $r = 1279$, $b = 1$, computes $x = 25898$, and sends this to B.
   (b) B sends to A the 3-bit vector $(0, 0, 1)$.
   (c) A computes and sends to B $y = r \cdot s_3 \bmod n = 403104$.
   (d) B computes $z = y^2 \cdot v_3 \bmod n = 25898$ and accepts A's identity since $z = +x$ and $z \ne 0$. □

10.28 Note (*security of Feige-Fiat-Shamir identification protocol*)

(i) *probability of forgery.* Protocol 10.26 is provably secure against chosen message attack in the following sense: provided that factoring n is difficult, the best attack has a probability 2^{-kt} of successful impersonation.
(ii) *security assumption required.* The security relies on the difficulty of extracting square roots modulo large composite integers n of unknown factorization. This is equivalent to that of factoring n (see Fact 3.46).
(iii) *zero-knowledge and soundness.* The protocol is, relative to a trusted server, a (sound) zero-knowledge proof of knowledge provided $k = O(\log \log n)$ and $t = \Theta(\log n)$. See Remark 10.34 regarding the practical significance of such constraints. A simplistic view for fixed k is that the verifier, interested in soundness, favors larger t (more iterations) for a decreased probability of fraud; while the prover, interested in zero-knowledge, favors smaller t.
(iv) *parameter selection.* Choosing k and t such that $kt = 20$ allows a 1 in a million chance of impersonation, which suffices in the case that an identification attempt requires a personal appearance by a would-be impersonator (see §10.5). Computation, memory, and communication can be traded off; $1 \le k \le 18$ was originally suggested as appropriate. Specific parameter choices might be, for security 2^{-20}: $k = 5, t = 4$; for 2^{-30}: $k = 6, t = 5$.
(v) *security trade-off.* Both computation and communication may be reduced by trading off security parameters to yield a single iteration ($t = 1$), holding the product kt constant and increasing k while decreasing t; however, in this case the protocol is no longer a zero-knowledge proof of knowledge.

10.29 Note (*modifications to Feige-Fiat-Shamir*)

(i) As an alternative to step 1 of Protocol 10.26, each user may pick its own such modulus n. T is still needed to associate each user with its modulus.

(ii) The communication complexity can be reduced if A sends B (e.g., 128 bits of) a hash value $h(x)$ instead of x in message (1), with B's verification modified accordingly.

(iii) The scheme can be made *identity-based* as follows (cf. §13.4.3). T assigns a distinguished identifying string I_A to each party A (e.g., A's name, address, or other information which a verifier may wish to corroborate). A's public values v_i, $1 \le i \le k$ are then derived by both T and other parties B as $v_i = f(I_A, i)$ using an appropriate function f. Then the trusted center, knowing the factorization of n, computes a square root s_i of each v_i and gives these to A.
As an example of f, consider, for a randomly chosen but known value c, $f(I_A, i) = I_A + i + c \bmod n$. Since a square root of $f_i = f(I_A, i)$ is required, any f_i with Jacobi symbol -1 mod n may be multiplied by a fixed number with Jacobi symbol -1. A non-residue f_i with Jacobi $+1$ may be either discarded (A must then indicate to B, e.g., in message (3), which values i allow computation of the v_j); or mapped to a residue via multiplication by -1, again with an indication to B of this to allow computation of v_j. Note that both cases for dealing with a non-residue f_i with Jacobi $+1$ reveal some (non-useful) information.

(iv) The *parallel version* of the protocol, in which each of three messages contains the respective data for all t rounds simultaneously, can be shown to be secure (it releases no "transferable information"), but for technical reasons loses the zero-knowledge property. Such parallel execution (as opposed to *sequential iteration*) in interactive proofs allows the probability of error (forgery) to be decreased without increasing the number of rounds.

10.30 Note (*converting identification to signature scheme*) The following general technique may be used to convert an identification scheme involving a witness-challenge-response sequence to a signature scheme: replace the random challenge e of the verifier by the one-way hash $e = h(x||m)$, of the concatenation of the witness x and the message m to be signed (h essentially plays the role of verifier). As this converts an interactive identification scheme to a non-interactive signature scheme, the bitsize of the challenge e must typically be increased to preclude off-line attacks on the hash function.

10.4.3 GQ identification protocol

The Guillou-Quisquater (GQ) identification scheme (Protocol 10.31) is an extension of the Fiat-Shamir protocol. It allows a reduction in both the number of messages exchanged and memory requirements for user secrets and, like Fiat-Shamir, is suitable for applications in which the claimant has limited power and memory. It involves three messages between a claimant A whose identity is to be corroborated, and a verifier B.

10.31 Protocol GQ identification protocol

SUMMARY: A proves its identity (via knowledge of s_A) to B in a 3-pass protocol.

1. *Selection of system parameters.*
   (a) An authority T, trusted by all parties with respect to binding identities to public keys, selects secret random RSA-like primes p and q yielding a modulus $n = pq$. (As for RSA, it must be computationally infeasible to factor n.)

   (b) T defines a public exponent $v \geq 3$ with $\gcd(v, \phi) = 1$ where $\phi = (p-1)(q-1)$, and computes its private exponent $s = v^{-1} \bmod \phi$. (See Note 10.33.)
   (c) System parameters (v, n) are made available (with guaranteed authenticity) for all users.
2. *Selection of per-user parameters.*
   (a) Each entity A is given a unique identity I_A, from which (the *redundant identity*) $J_A = f(I_A)$, satisfying $1 < J_A < n$, is derived using a known redundancy function f. (See Note 10.35. Assuming that factoring n is difficult implies $\gcd(J_A, \phi) = 1$.)
   (b) T gives to A the secret (*accreditation data*) $s_A = (J_A)^{-s} \bmod n$.
3. *Protocol messages.* Each of t rounds has three messages as follows (often $t = 1$).

$$\begin{array}{lll} A \rightarrow B: & I_A,\ x = r^v \bmod n & (1) \\ A \leftarrow B: & e \text{ (where } 1 \leq e \leq v) & (2) \\ A \rightarrow B: & y = r \cdot {s_A}^e \bmod n & (3) \end{array}$$

4. *Protocol actions.* A proves its identity to B by t executions of the following; B accepts the identity only if all t executions are successful.
   (a) A selects a random secret integer r (the *commitment*), $1 \leq r \leq n-1$, and computes (the *witness*) $x = r^v \bmod n$.
   (b) A sends to B the pair of integers (I_A, x).
   (c) B selects and sends to A a random integer e (the *challenge*), $1 \leq e \leq v$.
   (d) A computes and sends to B (the *response*) $y = r \cdot {s_A}^e \bmod n$.
   (e) B receives y, constructs J_A from I_A using f (see above), computes $z = {J_A}^e \cdot y^v \bmod n$, and accepts A's proof of identity if both $z = x$ and $z \neq 0$. (The latter precludes an adversary succeeding by choosing $r = 0$.)

10.32 Example (*GQ identification protocol with artificially small parameters and* $t = 1$)

1. (a) The authority T selects primes $p = 569$, $q = 739$, and computes $n = pq = 420491$.
   (b) T computes $\phi = (p-1)(q-1) = 419184$, selects $v = 54955$, and computes $s = v^{-1} \bmod \phi = 233875$.
   (c) System parameters $(54955, 420491)$ are made available for all users.
2. (a) Suppose that A's redundant identity is $J_A = 34579$.
   (b) T gives to A the accreditation data $s_A = (J_A)^{-s} \bmod n = 403154$.
3. See Protocol 10.31 for a summary of the messages exchanged.
4. (a) A selects $r = 65446$ and computes $x = r^v \bmod n = 89525$.
   (b) A sends to B the pair $(I_A, 89525)$.
   (c) B sends to A the random challenge $e = 38980$.
   (d) A sends $y = r \cdot {s_A}^e \bmod n = 83551$ to B.
   (e) B computes $z = {J_A}^e \cdot y^v \bmod n = 89525$ and accepts A's identity since $z = x$ and $z \neq 0$. □

10.33 Note (*security of GQ identification protocol*)

(i) *probability of forgery.* In Protocol 10.31, v determines the security level (cf. Fiat-Shamir where $v = 2$ but there are many rounds); some values such as $v = 2^{16}+1$ may offer computational advantages. A fraudulent claimant can defeat the protocol with a 1 in v chance by guessing e correctly a priori (and then forming $x = {J_A}^e \cdot y^v$ as the verifier would). The recommended bitlength of v thus depends on the environment under which attacks could be mounted (see §10.5).

(ii) *security assumption required.* Extracting v^{th} roots modulo the composite integer n (i.e., solving the RSA problem – §3.3) appears necessary to defeat the protocol; this is no harder than factoring n (Fact 3.30), and appears computationally intractable without knowing the factors of n.

(iii) *soundness.* In practice, GQ with $t = 1$ and a k-bit prime v is often suggested. For generalized parameters (n, v, t), the probability of forgery is v^{-t}. If v is constant, then technically for soundness, t must grow asymptotically faster than $\log \log n$. (For soundness, $v^{-t} = O(e^{-kt})$ must be smaller than inverse-polynomial in $\log n$; only polynomial security is provided if for a constant c, $v^t = O((\log n)^c)$. See also Remark 10.34.)

(iv) *zero-knowledge property.* In opposition to the soundness requirement, for GQ to be zero-knowledge apparently requires $tv = O((\log n)^c)$ for constant c, imposing an upper bound on t asymptotically: for v constant, t must be no larger than polynomial in $\log n$.

10.34 Remark (*asymptotic concepts vs. practical protocols*) The asymptotic conditions for soundness specified in Note 10.33 have little meaning in practice, e.g., because big-O notation is not applicable once fixed values are assigned to parameters. Indeed, zero-knowledge is a theoretical concept; while complexity-theoretic definitions offer guidance in selecting practical security parameters, their significance diminishes when parameters are fixed. Regarding Note 10.33, if $t = 1$ is viewed as the instantiation of a non-constant parameter (e.g., the iterated logarithm of n), then $t = 1$ will suffice for all practical purposes; consider $n = 1024$, $t = \lceil \lg^4 n \rceil = 1$.

10.35 Note (*redundancy function for identity-based GQ*)

(i) The protocol as given is an identity-based version (cf. Note 10.29), where A's public key is reconstructed from identifier I_A sent in message (1). Alternatively, a certified public key may be used, distributed in a certificate as per Protocol 10.36.

(ii) One example of the redundancy function f is the redundancy mapping of the preprocessing stage of ISO/IEC 9796 (see §11.3.5). A second example is a single function value of f as in Note 10.29, for an appropriate value i.

(iii) The purpose of the redundancy is to preclude an adversary computing false accreditation data corresponding to a plausible identity; this would be equivalent to forging a certificate in certificate-based schemes.

10.4.4 Schnorr identification protocol

The Schnorr identification protocol is an alternative to the Fiat-Shamir and GQ protocols. Its security is based on the intractability of the discrete logarithm problem. The design allows pre-computation, reducing the real-time computation for the claimant to one multiplication modulo a prime q; it is thus particularly suitable for claimants of limited computational ability. A further important computational efficiency results from the use of a subgroup of order q of the multiplicative group of integers modulo p, where $q|(p-1)$; this also reduces the required number of transmitted bits. Finally, the protocol was designed to require only three passes, and a low communications bandwidth (e.g., compared to Fiat-Shamir).

The basic idea is that A proves knowledge of a secret a (without revealing it) in a time-variant manner (depending on a challenge e), identifying A through the association of a with the public key v via A's authenticated certificate.

10.36 Protocol Schnorr identification protocol

SUMMARY: A proves its identity to B in a 3-pass protocol.

1. *Selection of system parameters.*
   (a) A suitable prime p is selected such that $p-1$ is divisible by another prime q. (Discrete logarithms modulo p must be computationally infeasible – see §3.6; e.g., $p \approx 2^{1024}$, $q \geq 2^{160}$.)
   (b) An element β is chosen, $1 \leq \beta \leq p-1$, having multiplicative order q. (For example, for α a generator mod p, $\beta = \alpha^{(p-1)/q} \bmod p$; see Note 4.81.)
   (c) Each party obtains an authentic copy of the system parameters (p, q, β) and the verification function (public key) of the trusted party T, allowing verification of T's signatures $S_T(m)$ on messages m. (S_T involves a suitable known hash function prior to signing, and may be any signature mechanism.)
   (d) A parameter t (e.g., $t \geq 40$), $2^t < q$, is chosen (defining a security level 2^t).
2. *Selection of per-user parameters.*
   (a) Each claimant A is given a unique identity I_A.
   (b) A chooses a private key a, $0 \leq a \leq q-1$, and computes $v = \beta^{-a} \bmod p$.
   (c) A identifies itself by conventional means (e.g., passport) to T, transfers v to T with integrity, and obtains a certificate $cert_A = (I_A,\ v,\ S_T(I_A, v))$ from T binding I_A with v.
3. *Protocol messages.* The protocol involves three messages.

$$\begin{array}{lll} A \to B: & cert_A,\ x = \beta^r \bmod p & (1) \\ A \leftarrow B: & e \text{ (where } 1 \leq e \leq 2^t < q) & (2) \\ A \to B: & y = ae + r \bmod q & (3) \end{array}$$

4. *Protocol actions.* A identifies itself to verifier B as follows.
   (a) A chooses a random r (the *commitment*), $1 \leq r \leq q-1$, computes (the *witness*) $x = \beta^r \bmod p$, and sends (1) to B.
   (b) B authenticates A's public key v by verifying T's signature on $cert_A$, then sends to A a (never previously used) random e (the *challenge*), $1 \leq e \leq 2^t$.
   (c) A checks $1 \leq e \leq 2^t$ and sends B (the *response*) $y = ae + r \bmod q$.
   (d) B computes $z = \beta^y v^e \bmod p$, and accepts A's identity provided $z = x$.

10.37 Example (*Schnorr identification protocol with artificially small parameters*)

1. (a) The prime $p = 48731$ is selected, where $p-1$ is divisible by the prime $q = 443$.
   (b) A generator mod 48731 is $\alpha = 6$; β is computed as $\alpha^{(p-1)/q} \bmod p = 11444$.
   (c) The system parameters are $(48731, 443, 11444)$.
   (d) The parameter $t = 8$ is chosen.
2. (b) A chooses a private key $a = 357$ and computes $v = \beta^{-a} \bmod p = 7355$.
3. See Protocol 10.36 for a summary of the messages exchanged.
4. (a) A chooses $r = 274$ and sends $x = \beta^r \bmod p = 37123$ to B.
   (b) B sends to A the random challenge $e = 129$.
   (c) A sends B the number $y = ae + r \bmod q = 255$.
   (d) B computes $z = \beta^y v^e \bmod p = 37123$ and accept's A's identity since $z = x$. □

10.38 Note (*security of Schnorr identification protocol*)

(i) *probability of forgery*. In Protocol 10.36, t must be sufficiently large to make the probability 2^{-t} of correctly guessing the challenge e negligible. $t = 40$, $q \geq 2^{2t} = 2^{80}$ was originally suggested in the case that a response is required within seconds (see §10.5); larger q may be necessary to preclude time-memory trade-offs, and $q \geq 2^{160}$ is recommended to preclude other off-line discrete log attacks. Correctly guessing e allows an adversary to impersonate A by choosing any y, sending $x = \beta^y v^e \bmod p$ to B in (1), then sending y in (3).

(ii) *soundness.* It can be shown that the protocol is a proof of knowledge of a, i.e., any party completing the protocol as A must be capable of computing a. Informally, the protocol reveals "no useful information" about a because x is a random number, and y is perturbed by the random number r. (However, this does not prove that adversarial discovery of a is difficult.)

(iii) *zero-knowledge property.* The protocol is not zero-knowledge for large e, because through interaction, B obtains the solution (x, y, e) to the equation $x = \beta^y v^e \bmod p$, which B itself might not be able to compute (e.g., if e were chosen to depend on x).

10.39 Note (*reducing transmission bandwidth*) The number of bits transmitted in the protocol can be reduced by replacing x in message (1) by t pre-specified bits of x (e.g., the least significant t bits), and having B compare this to t corresponding bits of z.

10.4.5 Comparison: Fiat-Shamir, GQ, and Schnorr

The protocols of Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr all provide solutions to the identification problem. Each has relative advantages and disadvantages with respect to various performance criteria and for specific applications. To compare the protocols, a typical set of selected parameters must be chosen for each providing comparable estimated security levels. The protocols may then be compared based on the following criteria:

1. *communications*: number of messages exchanged, and total bits transferred;
2. *computations*: number of modular multiplications for each of prover and verifier (noting on-line and off-line computations);
3. *memory*: storage requirements for secret keys (and signature size, in the case of signature schemes);
4. *security guarantees*: comparisons should consider security against forgery by guessing (soundness), possible disclosure of secret information (zero-knowledge property), and status regarding provable security; and
5. *trust required in third party*: variations of the protocols may require different trust assumptions in the trusted party involved.

The number of criteria and potential parameter choices precludes a comparison which is both definitive and concise. The following general comments may, however, be made.

1. *computational efficiency.* Fiat-Shamir requires between one and two orders of magnitude fewer full modular multiplications (steps) by the prover than an RSA private-key operation (cf. §10.3.3). When $kt = 20$ and n is 512 bits, Fiat-Shamir uses from about 11 to about 30 steps ($k = 20$, $t = 1$; and $k = 1$, $t = 20$); GQ requires about 60 steps (for $t = 1$, $m = 20 = \log_2(v)$), or somewhat fewer if v has low Hamming weight; and full exponentiation in unoptimized RSA takes 768 steps.

2. *off-line computations.* Schnorr identification has the advantage of requiring only a single on-line modular multiplication by the claimant, provided exponentiation may be done as a precomputation. (Such a trade-off of on-line for off-line computation is possible in some applications; in others, the total computation must be considered.) However, significant computation is required by the verifier compared to Fiat-Shamir and GQ.
3. *bandwidth and memory for secrets.* GQ allows the simultaneous reduction of both memory (parameter k) and transmission bandwidth (parameter t) with $k = t = 1$, by introducing the public exponent $v > 2$ with the intention that the probability of successful cheating becomes v^{-kt}; this simultaneous reduction is not possible in Fiat-Shamir, which requires k user secrets and t iterations for an estimated security (probability of cheating) of 2^{-kt}. Regarding other tradeoffs, see Note 10.28.
4. *security assumptions.* The protocols require the assumptions that the following underlying problems are intractable, for a composite (RSA) integer n: Fiat-Shamir – extracting square roots mod n; GQ – extracting v^{th} roots mod n (i.e., the RSA problem); Schnorr identification – computing discrete logs modulo a prime p.

10.5 Attacks on identification protocols

The methods an adversary may employ in an attempt to defeat identification protocols are a subset of those discussed in Chapter 12 for authenticated key establishment, and the types of adversaries may be similarly classified (e.g., passive vs. active, insider vs. outsider); for a discussion of attacks on simple password schemes, see §10.2.2. Identification is, however, less complex than authenticated key establishment, as there is no issue of an adversary learning a previous session key, or forcing an old key to be reused. For conciseness, the following definitions are made:

1. *impersonation:* a deception whereby one entity purports to be another.
2. *replay attack:* an impersonation or other deception involving use of information from a single previous protocol execution, on the same or a different verifier. For stored files, the analogue of a replay attack is a *restore* attack, whereby a file is replaced by an earlier version.
3. *interleaving attack:* an impersonation or other deception involving selective combination of information from one or more previous or simultaneously ongoing protocol executions (*parallel sessions*), including possible origination of one or more protocol executions by an adversary itself.
4. *reflection attack:* an interleaving attack involving sending information from an ongoing protocol execution back to the originator of such information.
5. *forced delay:* a forced delay occurs when an adversary intercepts a message (typically containing a sequence number), and relays it at some later point in time. Note the delayed message is not a replay.
6. *chosen-text attack:* an attack on a challenge-response protocol wherein an adversary strategically chooses challenges in an attempt to extract information about the claimant's long-term key.
   Chosen-text attacks are sometimes referred to as using the claimant as an *oracle*, i.e., to obtain information not computable from knowledge of a claimant's public key alone. The attack may involve chosen-plaintext if the claimant is required to sign,

encrypt, or MAC the challenge, or chosen-ciphertext if the requirement is to decrypt a challenge.

Potential threats to identification protocols include impersonation by any of the following attacks: replay, interleaving, reflection, or forced delay. Impersonation is also trivial if an adversary is able to discover an entity's long-term (secret or private) keying material, for example, using a chosen-text attack. This may be possible in protocols which are not zero-knowledge, because the claimant uses its private key to compute its response, and thus a response may reveal partial information. In the case of an active adversary, attacks may involve the adversary itself initiating one or more new protocol runs, and creating, injecting, or otherwise altering new or previous messages. Table 10.3 summarizes counter-measures for these attacks.

Type of attack	Principles to avoid attack
replay	use of challenge-response techniques; use of nonces; embed target identity in response
interleaving	linking together all messages from a protocol run (e.g., using chained nonces)
reflection	embed identifier of target party in challenge responses; construct protocols with each message of different form (avoid message symmetries); use of uni-directional keys
chosen-text	use of zero-knowledge techniques; embed in each challenge response a self-chosen random number (*confounder*)
forced delay	combined use of random numbers with short response time-outs; timestamps plus appropriate additional techniques

Table 10.3: *Identification protocol attacks and counter-measures.*

10.40 Remark (*use of keys for multiple purposes*) Caution is advised if any cryptographic key is used for more than one purpose. For example, using an RSA key for both entity authentication and signatures may compromise security by allowing a chosen-text attack. Suppose authentication here consists of B challenging A with a random number r_B RSA-encrypted under A's public key, and A is required to respond with the decrypted random number. If B challenges A with $r_B = h(x)$, A's response to this authentication request may (unwittingly) provide to B its RSA signature on the hash value of the (unknown to A) message x. See also Example 9.88, where a DES key used for both CBC encryption and CBC-MAC leads to a security flaw; and Remark 13.32.

10.41 Remark (*adversary acting "as a wire"*) In any identification protocol between A and B, an adversary C may step into the communications path and simply relay (without changing) the messages between legitimates parties A and B, itself acting as a part of the communications link. Typically in practice, this is not considered a true "attack", in the sense that it does not alter the aliveness assurance delivered by the protocol; however, in some special applications, this may be a concern (see Remark 10.42).

10.42 Remark (*grandmaster postal-chess problem*) Identification protocols do not provide assurances about the physical location of the authenticated party. Therefore, Remark 10.41 notwithstanding, a concern may arise in the special case that the following is possible: an adversary C attempts to impersonate B, is challenged (to prove it is B) by A, and is able to relay (in real time, without detection or noticeable delay, and pretending to be A) the challenge on to the real B, get a proper response from B, and pass this response along back to A. In this case, additional measures are necessary to prevent a challenged entity from eliciting aid in computing responses. This is related to the so-called *grandmaster postal-chess problem*, whereby an amateur's chess rating may unfairly be improved by engaging in two simultaneous chess games with distinct grandmasters, playing black in one game and white in the second, and using the grandmaster's moves from each game in the other. Either two draws, or a win and a loss, are guaranteed, both of which will improve the amateur's rating.

For further discussion of protocol attacks including specific examples of flawed entity authentication protocols, see §12.9.

(i) Maintaining authenticity

Identification protocols provide assurances corroborating the identity of an entity only at a given instant in time. If the continuity of such an assurance is required, additional techniques are necessary to counteract active adversaries. For example, if identification is carried out at the beginning of a communications session to grant communications permissions, a potential threat is an adversary who "cuts in" on the communications line immediately after the successful identification of the legitimate party. Approaches to prevent this include:

1. performing re-authentication periodically, or for each discrete resource requested (e.g., each file access). A remaining threat here is an adversary who "steps out" every time re-authentication is performed, allowing the legitimate party to perform this task, before re-entering.
2. tying the identification process to an ongoing integrity service. In this case, the identification process should be integrated with a key establishment mechanism, such that a by-product of successful identification is a session key appropriate for use in a subsequent ongoing integrity mechanism.

(ii) Security level required for on-line vs. off-line attacks

The security level required for identification protocols depends on the environment and the specific application at hand. The probability of success of "guessing attacks" should be considered, and distinguished from the amount of computation required to mount on-line or off-line attacks (using the best techniques known). Some illustrative notes follow (see also Note 10.28).

1. *Local attacks*. Selecting security parameters which limit the probability of successful impersonation of a guessing attack (an adversary simply guesses a legitimate party's secret) to a 1 in 2^{20} chance (20 bits of security) may suffice if, for each attempted impersonation, a local appearance is required by the would-be impersonator and there is a penalty for failed attempts. Depending on the potential loss resulting relative to the penalty, 10 to 30 bits or more of security may be required.
2. *Remote attacks*. A higher level of security is required in environments where unlimited identification attempts, each involving minimal computational effort, are possible by remote electronic communications, by an anonymous claimant interacting with an on-line system, with no penalties for failed attempts. 20 to 40 bits of security or more may be called for here, unless the number of interactions may be somehow limited.
3. *Off-line or non-interactive attacks*. Selecting security parameters such that an attack requires 2^{40} computations in real-time (during a protocol execution) may be acceptable, but a bound of 2^{60} to 2^{80} computations (the latter should be adequate in all

cases) may be called for if the computations can be carried out off-line, and the attack is *verifiable* (i.e., the adversary can confirm, before interacting with the on-line system, that his probability of successful impersonation is near 1; or can recover a long-term secret by off-line computations subsequent to an interaction).

10.6 Notes and further references

§10.1

Davies and Price [308] and Ford [414] provide extensive discussion of authentication and identification; see also the former for biometric techniques, as well as Everett [380]. The comprehensive survey on login protocols by de Waleffe and Quisquater [319] is highly recommended. Crépeau and Goutier provide a lucid concise summary of user identification techniques with Brassard [192]. For standardized entity authentication mechanisms, see ISO/IEC 9798 [598, 599, 600, 601, 602].

§10.2

See the §9.2 notes on page 377 for historical discussion of using a one-way function (*one-way cipher*) for "encrypted" password files. Morris and Thompson [907] introduce the notion of password salting in their 1979 report on UNIX passwords; in one study of 3289 user passwords unconstrained by password rules, 86% fell within an easily-searched subset of passwords. Feldmeier and Karn [391] give an update 10 years later, indicating 30% of passwords they encountered fell to their attack using a precomputed encrypted dictionary, sorted on tapes by salt values. See also Klein [680] and Lomas et al. [771]. Password salting is related to randomized encryption; the idea of padding plaintext with random bits before encryption may also be used to prevent *forward search* attacks on public-key encryption with small plaintext spaces. Password rules and procedures have been published by the U.S. Departments of Commerce [399] and Defense [334].

Methods for computing password-derived keys (§10.2.4) are specified in the Kerberos Authentication Service [1041] and PKCS #5 [1072]. A concern related to password-derived keys is that known plaintext allows password-guessing attacks; protocols specifically designed to prevent such attacks are mentioned in Chapter 12 notes on §12.6. The idea of chaining one-time passwords by a one-way function (Protocol 10.6) is due to Lamport [739]; for related practical applications, see RFC 1938 [1047]. Davies and Price [308, p.176] note a questionnaire-based identification technique related to fixed challenge-response tables, wherein the user is challenged by a random subset of previously answered questions.

§10.3

Needham and Schroeder [923] stimulated much early work in the area of authentication protocols in the late 1970s, and Needham was again involved with Burrows and Abadi [227] in the BAN logic work which stimulated considerable interest in protocol analysis beginning in the late 1980s; see Chapter 12 notes for further discussion.

Gong [501] provides an overview of both time variant parameters and message replay; see also Neuman and Stubblebine [925], and the annexes of parts of ISO/IEC 9798 (e.g., [600]). For security arguments against the use of timestamps and a discussion of implementation difficulties, see Bellovin and Merritt [103]; Gaarder and Snekkenes [433]; Diffie, van Oorschot, and Wiener [348]; and Gong [500], who considers postdated timestamps. See also §12.3 notes. Lam and Beth [734] note that timestamp-based protocols are appropriate for connectionless interactions whereas challenge-response suits connection-oriented communications, and suggest challenge-response techniques be used to securely synchronize timeclocks with applications themselves using timestamp-based authentication.

ISO/IEC 9798 [598] parts 2 through 5 specify entity authentication protocols respectively based on symmetric encryption [599], digital signatures [600], keyed one-way functions [601], and zero-knowledge techniques [602]; a subset of these are presented in this chapter. FIPS 196 [407] is a subset of 9798-3 containing the unilateral and mutual authentication protocols involving challenge-response with random numbers.

Several parts of 9798 were influenced by the SKID2 and SKID3 (*Secret Key IDentification*) protocols from the RACE/RIPE project [178], which leave the keyed hash function unspecified but recommend RIPE-MAC with 64-bit random-number challenges. Diffie [342, 345] notes that two-pass challenge-response identification based on encryption and random challenges has been used since the 1950s in military *Identification Friend or Foe* (IFF) systems to distinguish friendly from hostile aircraft. Mao and Boyd [781] discuss the danger of improperly using encryption in authentication protocols, specifically the CBC mode without an integrity mechanism (cf. Remark 10.16). Stubblebine and Gligor [1179] discuss attacks involving this same mode; see also the much earlier paper by Akl [20].

Davies and Price [308] give a concise discussion of password generators. The identification technique in §10.3.3(i) based on public-key decryption and witness is derived from a Danish contribution to the 4th Working Draft of ISO/IEC 9798-5, specifying a protocol called COMSET and motivated in part by Brandt et al. [188], and related to ideas noted earlier by Blum et al. [163].

§10.4

A refreshingly non-mathematical introduction to zero-knowledge proofs is provided by Quisquater, Guillou, and Berson [1020], who document the secret of Ali Baba's legendary cave, and its rediscovery by Mick Ali. Mitropoulos and Meijer [883] give an exceptionally readable and comprehensive survey (circa 1990) of interactive proofs and zero knowledge, with a focus on identification. Other overviews include Johnson [641]; Stinson [1178, Ch.13]; and Brassard, Chaum, and Crépeau [193] (or [192]) for a discussion of *minimum disclosure* proofs, based on *bit commitment* and the primitive of a *blob*. Brassard and Crépeau [195] provide a user-friendly discussion of various definitions of zero-knowledge, while Goldreich and Oren [475] examine properties and relationships between various definitions of ZK proof systems.

Rabin [1022] employed the idea of *cut-and-choose protocols* for cryptographic applications as early as 1978. While Babai (with Moran) [60, 61] independently developed a theory of randomized interactive proofs known as *Arthur-Merlin games* in an attempt to "formalize the notion of efficient provability by overwhelming statistical evidence", interactive proof systems and the notion of zero-knowledge (ZK) proofs were formalized in 1985 by Goldwasser, Micali, and Rackoff [481] in the context of an interactive *proof of membership* of a string x in a language $\mathcal{L}$; they showed that the languages of quadratic-residues and of quadratic non-residues each have ZK interactive proof (ZKIP) systems revealing only a single bit of knowledge, namely, that $x \in \mathcal{L}$. Goldreich, Micali, and Wigderson [473, 474] prove likewise for graph non-isomorphism (known not to be in **NP**) and graph isomorphism, and that assuming the existence of secure encryption schemes, every language in **NP** has a ZKIP; see also Chaum [244], and Brassard and Crépeau [194].

Motivated by cryptographic applications and identification in particular, Feige, Fiat, and Shamir [383] adapted the concepts of interactive proofs of membership to interactive *proofs* *of knowledge*, including reformulated definitions for completeness, soundness, and zero-knowledge; while proofs of membership reveal one bit of set membership information, proofs of knowledge reveal only one bit about the prover's *state* of knowledge. The definitions given in §10.4.1 are based on these. These authors refine the original scheme of Fiat and Shamir [395] to yield that of Protocol 10.26; both may be converted to identity-based schemes (Note 10.29) in the sense of Shamir [1115]. The Fiat-Shamir scheme is related to (but more efficient than) an earlier protocol for proving quadratic residuosity (presented at Eurocrypt'84, but unpublished) by Fischer, Micali, and Rackoff [412]. The Fiat-Shamir protocol as per Protocol 10.24 includes an improvement noted by Desmedt et al. [340] to avoid inverses in the derivation of user secrets; this optimization may also be made to Protocol 10.26.

Related to definitions in §10.4.1, Bellare and Goldreich [87] noted that Goldwasser, Micali, and Rackoff [481] did not formally propose a definition for a proof of knowledge, and suggested that the formal definitions of Feige, Fiat, and Shamir [383] and Tompa and Woll [1194] were unsatisfactory for some applications. To address these issues they proposed a new definition, having some common aspects with that of Feige and Shamir [384], but offering additional advantages.

Micali and Shamir [868] provide preliminary notes on reducing computation in the Fiat-Shamir protocol by choosing the public keys v_i, $1 \leq i \leq k$ to be the first k prime numbers; each user then has an independent modulus n. A modification of Fiat-Shamir identification by Ong and Schnorr [957] decreases computational complexity, signature size, and the number of communications required, condensing t Fiat-Shamir iterations into one iteration while leaving each user with k private keys (cf. the $k = 1$ extension below); for computational efficiency, they suggest using as secret keys (not too) small integers.

The idea of generalizing Fiat-Shamir identification in other ways, including "replacing square roots by cubic or higher roots", was suggested in the original paper; using higher roots allows users to reduce their number of private keys k, including to the limiting case $k = 1$. Guillou and Quisquater [524] proposed a specific formulation of this idea of "using deep coin tosses" as the GQ scheme (Protocol 10.31); apparently independently, Ohta and Okamoto [945, 944] proposed a similar formulation, including security analysis.

The Ohta-Okamoto (OO) version of this *extended Fiat-Shamir* scheme differs from the GQ version (Protocol 10.31) as follows: (1) in OO, rather than T computing s_A from identity I_A, A chooses its own secret $s_A \in \mathbb{Z}_n$ and publishes $I_A = {s_A}^v \bmod n$; and (2) the verification relation $x \equiv {J_A}^e \cdot y^v \pmod{n}$ becomes $y^v \equiv x \cdot {I_A}^e$. OO is more general in that, as originally proposed, it avoids the GQ (RSA) constraint that $\gcd(v, \phi(n)) = 1$. Subsequent analysis by Burmester and Desmedt [221] suggests that additional care may be required when v is not prime. While the OO version precludes an identity-based variation, a further subsequent version of extended Fiat-Shamir (GQ variation) by Okamoto [949] ("Scheme 3" of 5 protocols therein) is provably as secure as factoring, only slightly less efficient, and is amenable to an identity-based variation.

The zero-knowledge interactive protocols of Chaum et al. [248, 249] for proving possession of discrete logarithms, provided a basis for Protocol 10.36 which is due to Schnorr [1097, 1098]. Schnorr also proposed a preprocessing scheme to reduce real-time computation, but see de Rooij [314] regarding its security. The Schnorr identification and signature schemes must not both be used with the same parameters β, p [1098] (cf. Remark 10.40). Schnorr's protocol is related to the log-based identification scheme of Beth [123] also proven to be zero-knowledge. Burmester et al. [223] analyze (cf. Note 10.33) a generalized identification protocol encompassing all the well-known variations related to Fiat-Shamir and including those of both Chaum et al. and Beth noted above. Van de Graaf and Peralta [1200] give a ZK interactive protocol for proving that a Blum integer is a Blum integer.

Brickell and McCurley [207] propose a modification of Schnorr's identification scheme, in which q is kept secret and exponent computations are reduced modulo $p-1$ rather than q; it has provable security if factoring $p-1$ is difficult, and moreover security equivalent to that of Schnorr's scheme otherwise; a drawback is that almost 4 times as much computation is required by the claimant. Another variant of Schnorr's scheme by Girault [458, 461] was the first identity-based identification scheme based on discrete logs; it uses a composite modulus, and features the user choosing its own secret key, which remains unknown to the trusted party (cf. implicitly-certified public keys, §12.6.2). A further variation of Schnorr's identification protocol by Okamoto [949] ("Scheme 1") uses two elements β_1 and β_2, of order q, and is provably secure, assuming the computational infeasibility of computing the $\mathbb{Z}_p$ discrete logarithm $\log_{\beta_1} \beta_2$ of β_2 relative to β_1; it does, however, involve some additional computation.

Aside from the above protocols based on the computational intractability of the standard number-theoretic problems (factoring and discrete logarithms), a number of very efficient identification protocols have more recently been proposed based on **NP**-hard problems. Shamir [1116] proposed a zero-knowledge identification protocol based on the **NP**-hard *permuted kernel problem*: given an $m \times n$ matrix A over $\mathbb{Z}_p$, p prime (and relatively small, e.g., $p = 251$), and an n-vector V, find a permutation π on $\{1, \dots, n\}$ such that $V_\pi \in ker(A)$, where $ker(A)$ is the kernel of A consisting of all n-vectors W such that $AW = [0 \dots 0] \bmod p$. Patarin and Chauvaud [966] discuss attacks on the permuted kernel problem which are feasible for the smallest of parameter choices originally suggested, while earlier less efficient attacks are presented by Baritaud et al. [73] and Georgiades [447]. Stern [1176] proposed a practical zero-knowledge identification scheme based on the **NP**-hard *syndrome decoding* problem, following an earlier less practical scheme of Stern [1174] based on intractable problems in coding theory. Stern [1175] proposed another practical identification scheme based on an **NP**-hard combinatorial *constrained linear equations* problem, offering a very short key length, which is of particular interest in specific applications. Pointcheval [983] proposed another such scheme based on the **NP**-hard *perceptrons problem*: given an $m \times n$ matrix M with entries ± 1, find an n-vector y with entries ± 1 such that $My \geq 0$.

Goldreich and Krawczyk [469] pursue the fact that the original definition of ZK of Goldwasser, Micali, and Rackoff is not closed under sequential composition (this was noted earlier by D. Simon), establishing the importance of the stronger definitions of ZK formulated subsequently (e.g., *auxiliary-input* zero-knowledge – see Goldreich and Oren [475]), for which closure under sequential composition has been proven. They prove that even these strong formulations of ZK are not, however, closed under parallel composition (thus motivating the definition of weaker notions of zero-knowledge), and that 3-pass interactive ZK proofs of membership that are *black-box simulation ZK* exist only for languages in **BPP** (Definition 2.77); while the definition of "black-box simulation ZK" is more restrictive than the original definition of ZK, all known ZK protocols are ZK by this definition also. Consequently, protocols that are (formally) ZK are less practical than their corresponding 3-pass parallel versions.

As a replacement for the security requirement of zero knowledge in many protocols, Feige and Shamir [384] proposed *witness indistinguishability* and the related notion of *witness hiding protocols*. Unlike zero knowledge, witness indistinguishability is preserved under arbitrary composition of protocols.

Methods have been proposed to reduce the communication complexity of essentially all customized identification protocols, including the use of hash values in the first message (cf. Note 10.29; Note 10.39). Girault and Stern [462] examine the security implications of the length of such hash values, note that collision-resistance of the hash function suffices for the typically claimed security levels, and examine further optimizations of the communication complexity of such protocols, including use of *r-collision resistant* hash functions.

Blum, Feldman, and Micali [163] introduced the idea of non-interactive (or more clearly: *mono-directional*) ZK proofs, separating the notions of interactive proof systems and zero-knowledge protocols; here the prover and verifier share a random string, and communication is restricted to one-way (or the prover may simply publish a proof, for verification at some future time). De Santis, Micali, and Persiano [317] improve these results employing a weaker complexity assumption; Blum et al. [162] provide a summary and further improvements. While the technique of Remark 10.30, due to Fiat and Shamir [395], allows a zero-knowledge identification scheme to be converted to a signature scheme, the latter cannot be a sound zero-knowledge signature scheme because the very simulatability of the identification which establishes the ZK property would allow signature forgery (e.g., see Okamoto [949]).

A further flavor of zero-knowledge (cf. Definition 10.22) is *statistical* (or *almost perfect*) zero-knowledge; here the probability distributions of the transcripts must be *statistically indistinguishable* (indistinguishable by an examiner with unlimited computing power but given only polynomially many samples). Pursuing other characterizations, interactive protocols in which the assurance a verifier obtains is based on some unproven assumption may be distinguished as *arguments* (see Brassard and Crépeau [195]), with *proofs* then required to be free of any unproven assumptions, although possibly probabilistic.

For performance comparisons and tradeoffs for the Fiat-Shamir, Guillou-Quisquater, and Schnorr schemes, see Fiat and Shamir [395], Schnorr [1098], Okamoto [949], and Lim and Lee [768], among others. For an overview of chipcard technology and the use thereof for identification, see Guillou, Ugon, and Quisquater [527]; an earlier paper on chipcards is by Guillou and Ugon [526]. Knobloch [681] describes a preliminary chipcard implementation of the Fiat-Shamir protocol.

§10.5

Bauspiess and Knobloch [78] discuss issues related to Remark 10.41, including taking over a communications line after entity authentication has completed. Bengio et al. [113] discuss implementation issues related to identification schemes such as the Fiat-Shamir protocol, including Remark 10.42. Classes of replay attacks are discussed in several papers, e.g., see Syverson [1182] and the ISO/IEC 10181-2 authentication framework [610]. For further references on the analysis of entity authentication protocols and attacks, see the §12.9 notes.