

# Unit IV: User Authentication

(5 Hours) — Cryptography and Network Security

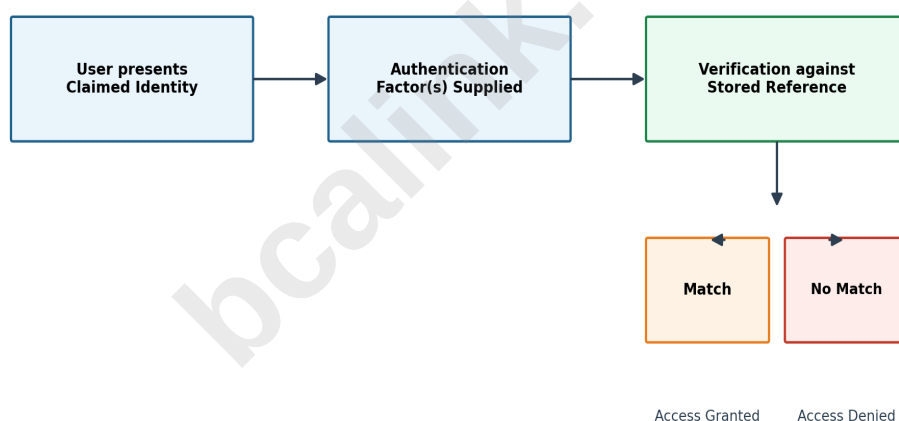
## 1. User Authentication Principles

### 1.1 What is User Authentication?

User authentication is the process of verifying that a person (or a process acting on behalf of a person) really is who they claim to be, before granting them access to a system or resource. In plain language: it is the digital equivalent of a security guard checking your ID card before letting you into a building. It always happens in two stages — first the user makes an identity claim (e.g. typing a username), and then the system verifies that claim using one or more authentication factors.

**Figure: General Model of User Authentication**

*Two distinct steps: Identification (who you claim to be) and Verification (proving that claim using one or more authentication factors).*



*Figure 1.1 — General Model of User Authentication*

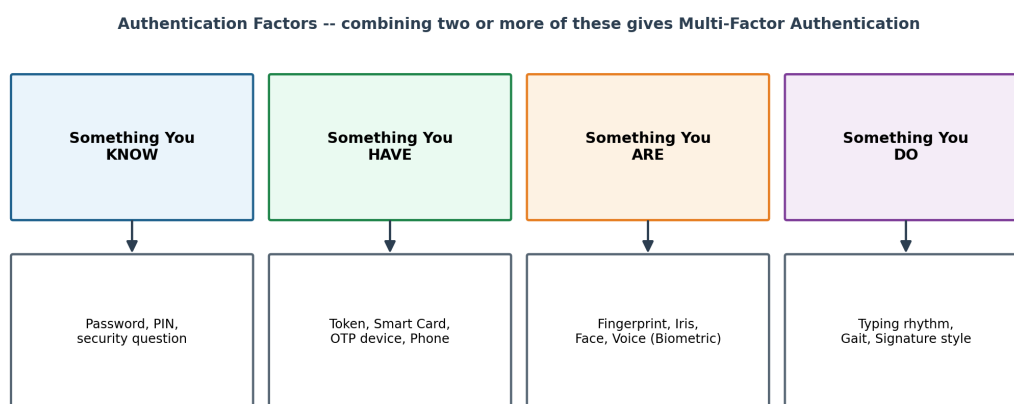
### 1.2 The Four Basic Means of Authentication

Security textbooks group every authentication technique into four broad categories, often called authentication factors:

1. Something the individual knows — a password, PIN, or answer to a secret question.
2. Something the individual possesses (a token) — a smart card, hardware token, or one-time-password generating app on a phone.
3. Something the individual is (static biometrics) — fingerprint, iris pattern, facial geometry, or voice print.

4. Something the individual does (dynamic biometrics) — typing rhythm, voice pattern while speaking, signature style, or gait while walking.

**Figure: The Four Basic Means of User Authentication**



*Figure 1.2 — The Four Basic Means of User Authentication*

### 1.3 Identification vs Verification vs Authorization

| Term                          | Plain-English Meaning                                                                 |
|-------------------------------|---------------------------------------------------------------------------------------|
| Identification                | Claiming an identity, e.g. typing a username or scanning an ID card.                  |
| Verification / Authentication | Proving that claimed identity is genuine, using a factor (password, OTP, biometric).  |
| Authorization                 | Deciding what the now-verified user is actually allowed to do (separate, later step). |

### 1.4 Key Risks in Authentication Design

A well-designed authentication system must resist several common attack patterns: guessing or brute-forcing the credential, intercepting (eavesdropping) the credential in transit, replaying a previously captured credential, and a client-side or server-side database compromise revealing stored credentials in bulk. Designers reduce these risks using strong hashing, salting, encryption in transit (e.g. TLS), rate-limiting login attempts, and combining multiple independent factors.

*Note: No single authentication factor is perfect on its own — passwords can be guessed or phished, tokens can be lost or stolen, and even biometrics can sometimes be spoofed. This is exactly why Two-Factor and Multi-Factor Authentication (Section 6) combine independent factors together.*

## 2. Password-Based Authentication

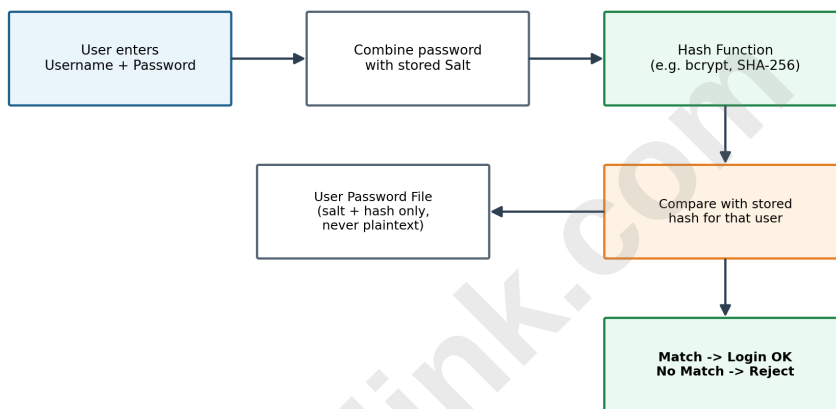
### 2.1 The Basic Idea

Password-based authentication is the oldest and still most common scheme: the user proves their identity simply by typing a secret string that only they (in theory) know. Its huge advantage is simplicity — no special hardware is required, and users already understand the concept.

## 2.2 How Passwords are Stored Safely

A system should never store passwords in plain text, because anyone who steals the password file (database) would instantly read every user's password. Instead, the system stores a salted hash of the password. A 'salt' is a random string unique to each user, combined with the password before hashing; this means even two users with the identical password end up with completely different stored hash values, and defeats precomputed 'rainbow table' attacks.

Figure: Secure Password Storage and Verification using Salted Hashing



Storing only a salted hash (never the plaintext password) protects user passwords even if the password file is stolen, and the salt defeats precomputed rainbow-table attacks.

Figure 2.1 — Secure Password Storage and Verification using Salted Hashing

### Example 2.1 — Why salting matters (illustrative)

Two users, Alice and Bob, both unfortunately choose the same password "sunshine". Using a simplified (illustrative) hash  $H(x) = \text{sum of ASCII codes of } x \text{ mod } 1000$ , show how salting produces different stored values for each, even with an identical password. (Assume Alice's salt = "7f3" and Bob's salt = "q2k", and for simplicity take the salted input as salt + password.)

### Solution:

Without salting, both Alice and Bob would hash the exact same string "sunshine", giving an identical stored hash — an attacker who cracks one password instantly knows the other.

With salting: Alice's salted input = "7f3sunshine", Bob's salted input = "q2ksunshine". These are different input strings, so even with the same simplified mod-hash function, they will almost certainly produce different output digests.

This demonstrates the core benefit of salting: identical passwords no longer produce identical stored hashes, which also defeats precomputed dictionary/rainbow-table attacks that rely on matching common password hashes directly.

## 2.3 Common Password Attacks

| Attack Type          | Plain-English Description                                                           |
|----------------------|-------------------------------------------------------------------------------------|
| Brute-force attack   | Systematically trying every possible password combination.                          |
| Dictionary attack    | Trying common words and known leaked passwords instead of all combinations.         |
| Rainbow table attack | Using precomputed hash-to-password lookup tables to reverse a stolen hash quickly.  |
| Phishing             | Tricking the user into typing their password into a fake, attacker-controlled site. |
| Credential stuffing  | Reusing passwords leaked from one breached site to log into other sites.            |
| Shoulder surfing     | Simply watching the user type their password in person.                             |

## 2.4 Good Password Policy Practices

1. Enforce a reasonable minimum length and complexity, without forcing impossible-to-remember rules.
2. Use a strong, slow hashing algorithm designed for passwords (e.g. bcrypt, scrypt, Argon2) rather than a fast general-purpose hash.
3. Always combine with a unique salt per user.
4. Limit and slow down repeated failed login attempts (rate limiting / account lockout).
5. Encourage or require password managers and discourage password reuse across sites.

*Note: Modern security guidance (e.g. NIST) has shifted away from forcing frequent mandatory password changes, since this often leads users to choose weaker, more predictable passwords. Length and uniqueness matter more than complexity rules alone.*

## 3. Token-Based Authentication

### 3.1 What is a Token?

A token is a physical or software-based device that the user possesses and that generates or stores a piece of authentication evidence, such as a One-Time Password (OTP), a cryptographic key, or a digital certificate. Because it represents 'something you have', it is fundamentally different from a password ('something you know'), and is usually combined with a password for stronger security.

### 3.2 Types of Tokens

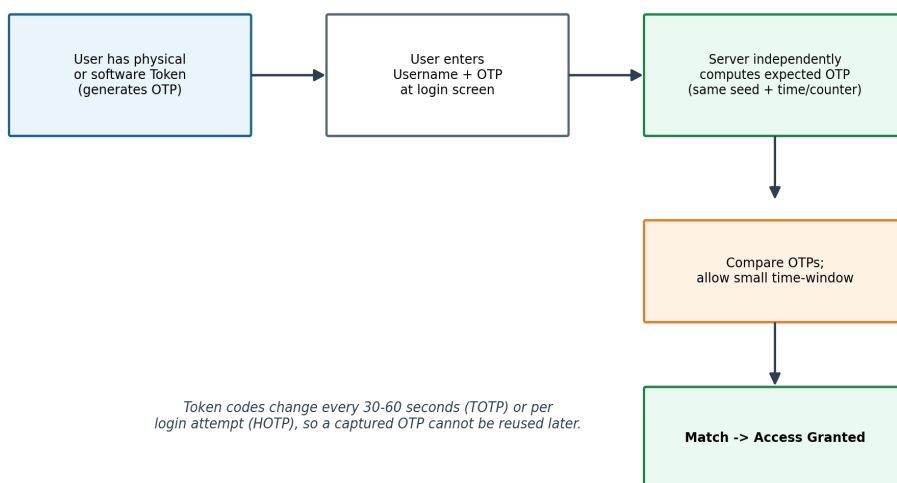
| Token Type          | Plain-English Description                                                                 |
|---------------------|-------------------------------------------------------------------------------------------|
| Hardware OTP token  | A small physical device (e.g. key fob) that displays a changing numeric code.             |
| Software/soft token | A mobile app (e.g. Authenticator app) that generates OTP codes on a smartphone.           |
| Smart card          | A card with an embedded chip holding cryptographic keys/certificates, used with a reader. |
| USB security key    | A small USB/NFC device implementing public-key based authentication (e.g. FIDO/U2F).      |
| SMS-based token     | A one-time code sent via text message (convenient, but weaker against SIM-swap attacks).  |

### 3.3 HOTP vs TOTP

Most OTP tokens follow one of two standard algorithms:

- HOTP (HMAC-based OTP): the code changes based on an internal event counter, incremented each time a new code is generated/used.
- TOTP (Time-based OTP): the code changes automatically based on the current time (typically a new code every 30 or 60 seconds), and is the most widely used scheme today (e.g. Google Authenticator, Microsoft Authenticator).

**Figure: Token-Based (One-Time-Password) Authentication**



*Figure 3.1 — Token-Based (One-Time-Password) Authentication*

**Example 3.1 — Simplified TOTP-style calculation**

Using a simplified (illustrative, non-cryptographic) formula  $OTP = (Secret\_Seed + Current\_Time\_Step) \bmod 1000000$ , where  $Secret\_Seed = 482913$  and the current 30-second  $Time\_Step$  counter value is 57. Calculate the 6-digit OTP the token would display.

**Solution:**

$$OTP = (482913 + 57) \bmod 1000000$$

$$OTP = 482970 \bmod 1000000$$

$$OTP = 482970$$

Displayed code (zero-padded to 6 digits) = 482970.

Thirty seconds later, the  $Time\_Step$  counter increments to 58, giving a completely different  $OTP = 482971$ 's worth, demonstrating how the code automatically changes at fixed intervals without any network communication needed by the token device itself.

*Note: Because the server independently computes the same expected OTP using the shared seed and current time/counter, no secret ever needs to travel over the network in the clear during normal operation — only the resulting short-lived numeric code is transmitted at login.*

**4. Biometric Authentication**

**4.1 What is Biometric Authentication?**

Biometric authentication verifies identity using measurable physical or behavioural characteristics of a person — 'something you are' or 'something you do'. Unlike a password, a biometric trait cannot be forgotten and is difficult to simply hand over to someone else, but it also cannot be 'changed' if it is ever compromised, which is an important design consideration.

**4.2 Common Biometric Modalities**

| Modality           | Plain-English Description                                                            |
|--------------------|--------------------------------------------------------------------------------------|
| Fingerprint        | Pattern of ridges and valleys on a fingertip; widely used on smartphones.            |
| Iris/Retina scan   | Unique patterns in the eye; very high accuracy, used in high-security settings.      |
| Face recognition   | Geometric/feature mapping of facial structure; common on phones and at airports.     |
| Voice recognition  | Unique vocal characteristics in pitch, tone, and speech patterns.                    |
| Signature dynamics | Not just the shape of a signature but the speed, pressure, and rhythm of writing it. |
| Keystroke dynamics | Unique typing rhythm and timing pattern of an individual on a keyboard.              |

### 4.3 Enrollment and Verification Pipeline

Every biometric system works in two phases. During enrollment, a sample is captured once and a compact mathematical 'feature template' is extracted and stored securely (never the raw image itself, ideally). During verification, a fresh sample is captured at login time, its template is extracted the same way, and a similarity (match) score is computed against the stored template.

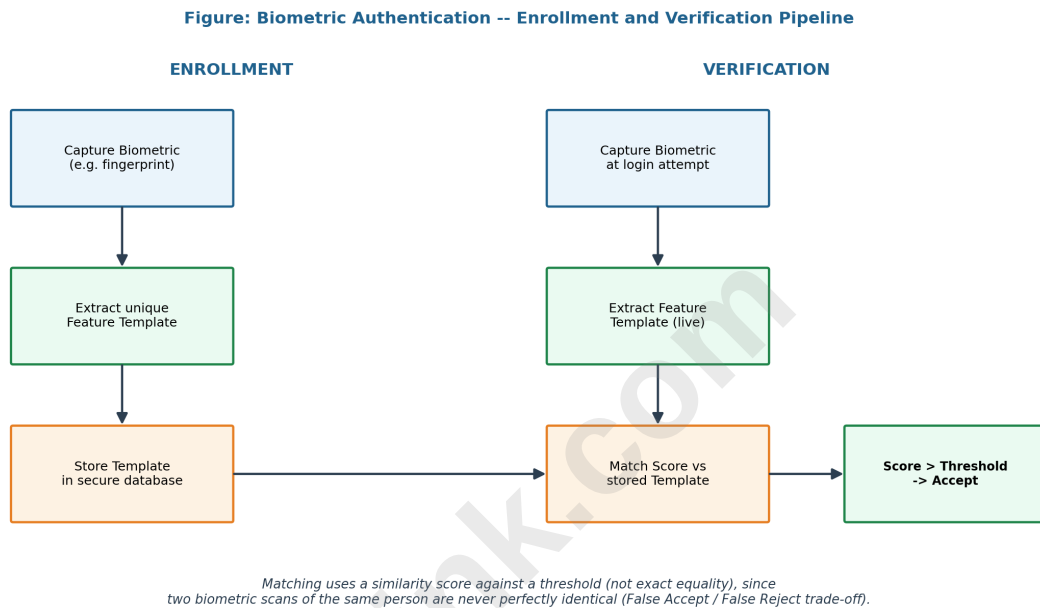


Figure 4.1 — Biometric Authentication Enrollment and Verification Pipeline

### 4.4 Accuracy Metrics: FAR and FRR

Biometric matching is probabilistic, not exact, so two error rates are used to measure performance:

- False Acceptance Rate (FAR): the percentage of times an impostor is incorrectly accepted as a genuine user.
- False Rejection Rate (FRR): the percentage of times a genuine user is incorrectly rejected.

There is a direct trade-off between the two: making the matching threshold stricter lowers FAR but raises FRR, and vice versa. The point where FAR equals FRR is called the Equal Error Rate (EER), often used as a single summary measure of a biometric system's accuracy.

#### Example 4.1 — Computing FAR and FRR from a small test set

A fingerprint system is tested with 500 genuine login attempts and 500 impostor attempts. The system incorrectly accepts 10 of the impostor attempts, and incorrectly rejects 15 of the genuine attempts. Compute the FAR and FRR.

**Solution:**

$\text{FAR} = (\text{Number of impostor attempts wrongly accepted} / \text{Total impostor attempts}) \times 100$

$\text{FAR} = (10 / 500) \times 100 = 2\%$ .

$\text{FRR} = (\text{Number of genuine attempts wrongly rejected} / \text{Total genuine attempts}) \times 100$

$\text{FRR} = (15 / 500) \times 100 = 3\%$ .

A FAR of 2% means roughly 2 in every 100 impostor attempts would be wrongly let in, while an FRR of 3% means roughly 3 in every 100 genuine users would be wrongly turned away and need to retry; tuning the match threshold could lower one of these values, but only at the cost of raising the other.

*Note: Because a stolen biometric trait cannot be 'reset' like a password, well-designed systems store only encrypted, non-reversible templates, and many also incorporate 'liveness detection' to resist spoofing attempts using a photo, recording, or fake fingerprint mould.*

## 5. Remote User Authentication

### 5.1 The Special Problem of Remote Authentication

When a user authenticates over an open, potentially insecure network (such as the internet), simply sending a password directly is dangerous — it could be intercepted (eavesdropped) by an attacker monitoring the network, or a captured copy could be replayed later to impersonate the user. Remote user authentication protocols are specifically designed to prove identity across such an untrusted channel without ever exposing the underlying secret in transit.

### 5.2 Challenge-Response Authentication

The most common solution is a challenge-response protocol. Instead of sending the password itself, the server sends a random, one-time challenge value (a 'nonce'). The client combines this challenge with its secret (e.g. by hashing them together) and sends back only the computed response. The server, who also knows the secret, performs the same computation and checks for a match. Since the challenge is different every single time, a captured response cannot be reused later (defeats replay attacks).

Figure: Remote User Authentication via Challenge-Response Protocol



Figure 5.1 — Remote User Authentication via Challenge-Response Protocol

### Example 5.1 — Challenge-response walkthrough

Using a simplified hash  $H(x) = \text{sum of ASCII codes of } x \text{ mod } 10000$ , a client and server share a secret  $S = \text{"key99"}$ . The server issues a random nonce  $N = \text{"482"}$ . Compute the response the client should send, and show how the server verifies it.

#### Solution:

Client combines challenge and secret: input =  $N + S = \text{"482key99"}$ .

ASCII sum: '4'=52, '8'=56, '2'=50, 'k'=107, 'e'=101, 'y'=121, '9'=57, '9'=57

Sum =  $52+56+50+107+101+121+57+57 = 601$ .

Response =  $601 \text{ mod } 10000 = 601$ .

The client sends Response = 601 back to the server (never sending the secret  $S$  itself over the network). The server, who also knows  $S = \text{"key99"}$ , independently computes the same input string and the same sum, getting 601, which matches — authentication succeeds.

Crucially, even if an attacker captures Response = 601, it is only valid for nonce  $N = 482$ ; the very next login will use a completely different random nonce, making the captured response useless for a replay attack.

## 5.3 Other Remote Authentication Approaches

- Kerberos: a trusted third-party ticket-granting authentication protocol widely used inside enterprise networks.
- Certificate-based (PKI) authentication: the client proves possession of a private key matching a trusted digital certificate.
- Federated/SSO authentication (e.g. SAML, OAuth/OpenID Connect): a trusted identity provider authenticates the user once and issues a token accepted by multiple remote services.

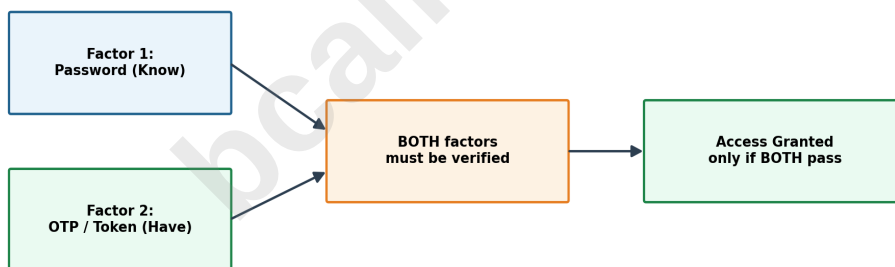
*Note: Always combine remote authentication with a secure transport channel such as TLS/HTTPS. Challenge-response protects the secret itself from exposure, but TLS additionally protects the confidentiality of the entire conversation, including usernames and session data.*

## 6. Two-Factor Authentication (2FA)

### 6.1 What is Two-Factor Authentication?

Two-Factor Authentication (2FA) requires a user to present two different, independent authentication factors (drawn from two different categories among 'know', 'have', 'are', 'do') before access is granted. This is much stronger than using two pieces of evidence from the SAME category (e.g. two passwords), because an attacker who compromises one type of factor still cannot succeed without separately compromising a completely different type of factor.

Figure: Two-Factor Authentication (2FA) Flow



*Even if a password is stolen (e.g. through phishing), the attacker still cannot log in without also possessing the second factor (e.g. the user's physical phone for the OTP). This is why 2FA dramatically reduces the success rate of credential-theft attacks.*

Figure 6.1 — Two-Factor Authentication (2FA) Flow

### 6.2 Common 2FA Combinations

| Factor 1        | Factor 2            | Real-World Example                                 |
|-----------------|---------------------|----------------------------------------------------|
| Password (Know) | OTP app code (Have) | Most online banking and email login flows today.   |
| Password (Know) | SMS code (Have)     | Common but weaker, vulnerable to SIM-swap attacks. |

| Factor 1        | Factor 2                     | Real-World Example                                  |
|-----------------|------------------------------|-----------------------------------------------------|
| PIN (Know)      | Fingerprint (Are)            | Smartphone unlock combining PIN fallback with bio   |
| Password (Know) | Hardware security key (Have) | High-security accounts (e.g. FIDO/U2F-based logins) |

### 6.3 Two-Factor vs Multi-Factor Authentication

Two-Factor Authentication is technically a specific case of the broader concept Multi-Factor Authentication (MFA), which simply means 'two or more independent factors'. Some very high-security systems use three factors (e.g. password + hardware token + fingerprint) for extremely sensitive operations, though 2FA is by far the most common balance of security and user convenience today.

#### Example 6.1 — Estimating attacker success probability with 2FA

Suppose an attacker has, on average, a 1 in 1000 (0.001) chance of successfully guessing/phishing a user's password, and independently a 1 in 500 (0.002) chance of also obtaining the user's OTP code for the same login attempt. Assuming the two events are independent, what is the probability the attacker successfully defeats BOTH factors?

#### Solution:

Since the two factors are independent, the combined probability is the **PRODUCT** of the individual probabilities:  $P(\text{both}) = P(\text{password}) \times P(\text{OTP})$ .

$P(\text{both}) = 0.001 \times 0.002 = 0.000002$ , i.e. 2 in 1,000,000.

Compare this to relying on the password alone, where the attacker's success probability would simply be 0.001 (1 in 1000). Adding the second independent factor reduces the attacker's overall success probability by a factor of 500 in this example, illustrating numerically why 2FA is so much stronger than relying on a single authentication factor.

*Note: 2FA is most effective when the two factors are truly independent — for example, an SMS code sent to the same phone used to read a phishing link offers weaker protection than an offline hardware token, since a single compromised device could potentially expose both factors at once.*

## Key Takeaways

- User authentication verifies a claimed identity using one or more of four factor categories: know, have, are, do.
- Passwords must always be stored as a salted hash, never in plain text, to resist theft and rainbow-table attacks.
- Token-based authentication ('something you have') typically uses HOTP (counter-based) or TOTP (time-based) one-time codes.
- Biometric authentication matches a live sample against a stored template using a similarity score, balancing FAR against FRR.
- Remote authentication over open networks uses challenge-response protocols so secrets are never transmitted directly and replay attacks are prevented.
- Two-Factor Authentication combines two independent factor categories, making an attacker's overall success probability the PRODUCT (and therefore much smaller) of each individual factor's weakness.

— *End of Unit IV* —