

Security, privacy and system integration

Protecting information and trusting communication
C / Arduino: integrating a complete system

Prof. Dr. Mehmet Göktürk · Gebze Technical University

A connected room sensor



Conceptual illustration. Where could someone read, change or interrupt the data?

What needs protection?

The reading is only one asset. Device settings, account credentials and the history of observations also matter.

A system can fail even when every transmitted byte is encrypted.

Begin with what must remain private, what must remain correct and what must remain available.

Security objectives

Objective	Meaning	Room-sensor example
Confidentiality	Restrict disclosure	Only authorized users see room history
Integrity	Detect or prevent unauthorized changes	A reading or threshold is not silently altered
Availability	Keep service usable when needed	Measurements and alerts remain accessible

The same incident can affect more than one objective.

Threat, vulnerability and risk

Term	Example
Threat	Someone attempts to change the alert threshold
Vulnerability	The configuration endpoint accepts changes without authorization
Risk	Consider the likelihood and consequences in this setting
Control	Require permission, validate changes and record an audit event

A threat model states what we protect, against whom, and under which assumptions.

Data arriving from another component must satisfy an explicit contract before it is used.

A local cable does not prove the sender is trusted. A well-formed message does not prove who sent it.

Identify where data crosses a device, process, account or network boundary.

Common attack mechanisms

Mechanism	What the attacker tries to do
Malware	Run unwanted code on a device
Phishing	Trick a person into disclosing information or approving an action
Eavesdropping / tampering	Read messages or alter their contents
Denial of service	Exhaust resources or otherwise prevent legitimate use

These are mechanisms, not mutually exclusive categories.

A convincing message can still be false

“Your account will close today. Sign in using this link.” Urgency is part of the attempt to influence behavior.

A familiar logo or sender display name is not evidence that the request is genuine.

For an unexpected request, reach the service through an independently known address and verify the action.

Authentication and authorization

Question	Operation	Example
Who controls this credential?	Authentication	Verify an account login
What may this account do?	Authorization	Allow viewing readings but forbid changing thresholds
What happened?	Audit logging	Record an approved configuration change

A successful login does not grant every permission.

Least privilege and layered protection

Give each component only the permissions needed for its task. A logger usually does not need administrator rights.

Physical access, software updates, access control and backups address different failure paths.

A backup supports recovery only if it can actually be restored. Encryption alone does not provide availability.

A password is a secret used in authentication. Reuse lets a compromise at one service threaten another.

A second factor adds a different kind of evidence, such as possession of an authenticator. Two passwords are still one factor type.

Phishing-resistant authentication binds an exchange to the legitimate service; not every two-factor method does this.

Store a salt and the output of a suitable password-hashing scheme, not a recoverable copy of the password.

The salt is not a secret. It makes equal passwords produce different stored results when salts differ.

A work factor makes each guess more expensive. A fast general-purpose hash alone is not a suitable password-storage design.

An encryption algorithm transforms plaintext into ciphertext using a key. Decryption recovers the plaintext with the appropriate key.

The algorithm can be public. Security depends on sound design, keys and correct use.

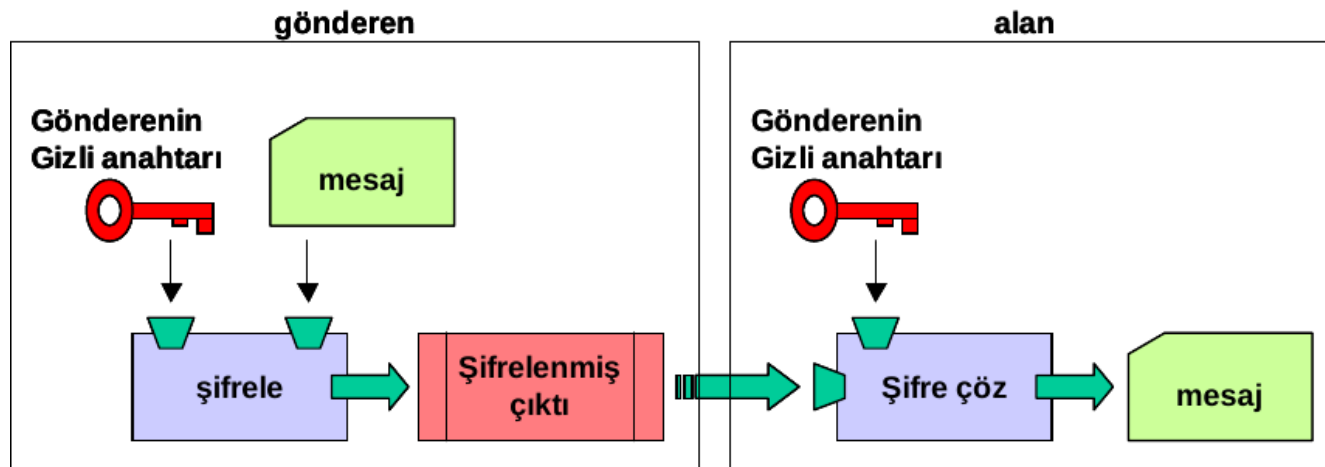
Encoding changes representation. Base64, for example, provides no secrecy.

A small cipher we can reverse

Step	Example with A=0, ..., Z=25
Plaintext	T E A $\rightarrow$ 19, 4, 0
Encrypt: $C = (P + 3) \bmod 26$	22, 7, 3 $\rightarrow$ W H D
Decrypt: $P = (C - 3) \bmod 26$	19, 4, 0 $\rightarrow$ T E A

A shift cipher is a classroom example, not secure encryption. Trying 26 shifts is easy.

Symmetric encryption



One shared secret protects this communication.

How do the endpoints obtain the secret safely?

Encryption alone does not necessarily detect tampering.

Both endpoints need the same secret key. Anyone who obtains it can use it.

AES is a standardized block cipher. A complete protocol also specifies how the cipher is used and how keys and nonces are handled.

Authenticated encryption protects confidentiality and detects tampering with the protected message.

DES means Data Encryption Standard. It is historical and unsuitable for new protection.

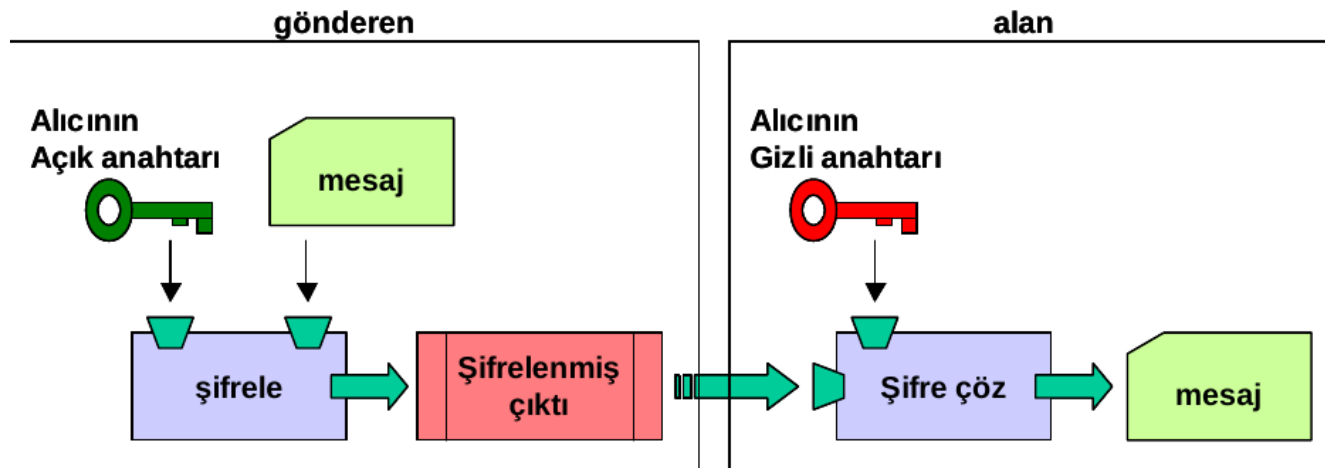
The shared-key distribution problem

Situation	Consequence
Two endpoints	They must establish and protect a shared secret
Five users, separate key for every pair	$5 \times 4 / 2 = 10$ pairwise keys
One group key for all users	Any member can act using that shared key
A member leaves the group	Old access may persist unless the relevant keys change

This simplified pairwise model motivates key management; it is not the only architecture.

Public-key encryption

The public key can be distributed.

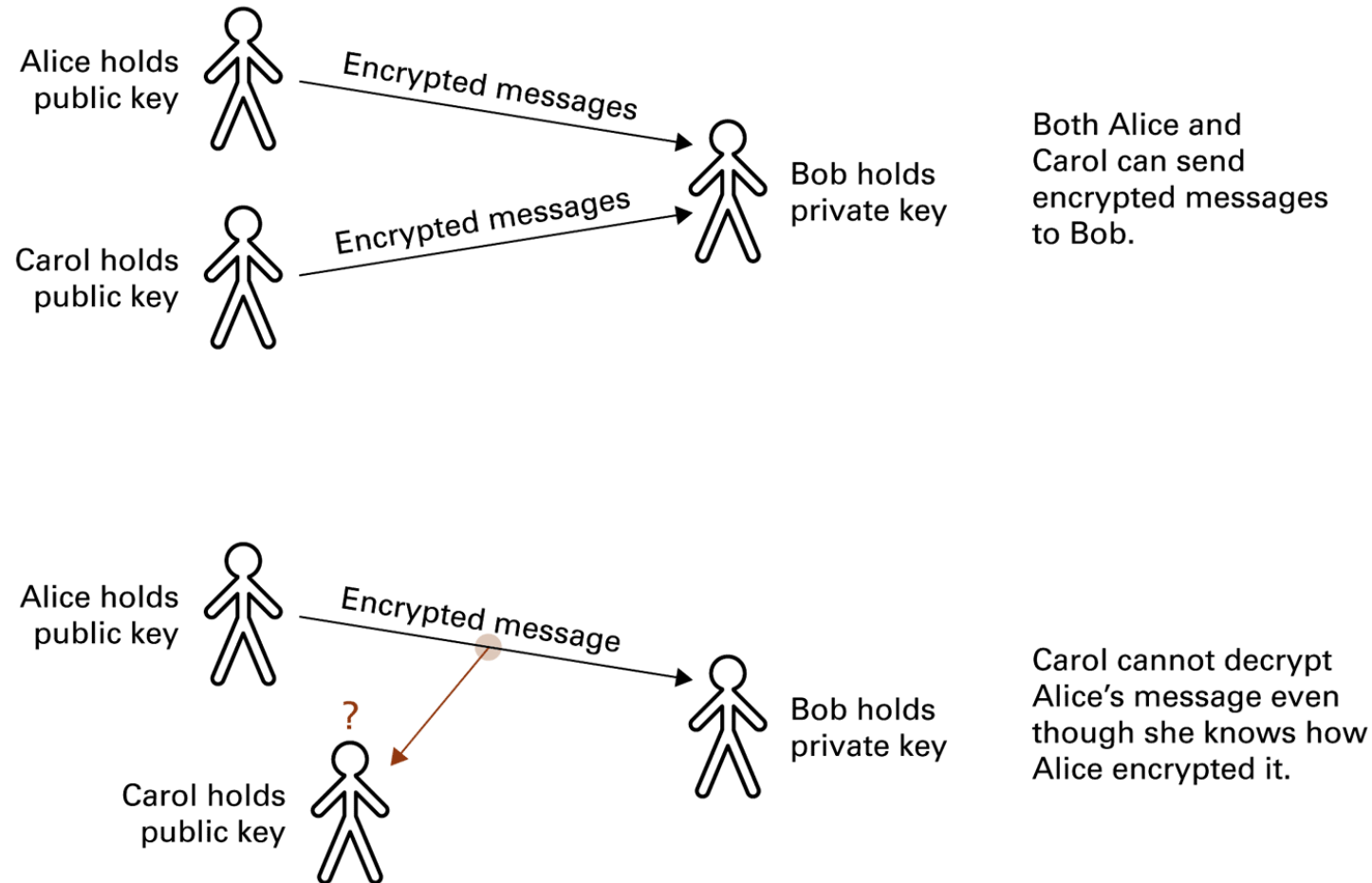


The recipient protects the private key.

We still need to know whose public key it is.

Encrypt to the recipient's public key; decrypt with the corresponding private key.

Who can read the message?



Knowing Bob's public key lets Alice and Carol encrypt to Bob. It does not let Carol decrypt Alice's message.

Public keys solve only part of the problem

Encrypting to Bob's public key protects the message for Bob. It does not prove that Alice sent it.

If an impostor substitutes a public key, a sender may encrypt to the impostor.

Protocols therefore combine key establishment with authentication and checks on key ownership.

Encryption, hashing and signatures

Operation	Main purpose	Secret involved
Encryption	Confidentiality; recover plaintext with the right key	Decryption key
Cryptographic hash	Compute a fixed-length digest	No secret for an ordinary hash
Digital signature	Verify signed data against a public key	Signer's private key

These operations answer different questions. One operation does not automatically provide the others.

A hash maps a message to a digest of fixed length. SHA-256 produces 256 bits.

A secure hash makes finding a matching preimage or a collision computationally difficult. Collisions still exist mathematically.

An ordinary hash has no secret key and no decryption operation.

One changed byte, a different digest

Exact UTF-8 input, no newline	First 16 hexadecimal digits of SHA-256
SET,600	4bcec4fc80e2fafc
SET,601	b20cc6738b7f73d9

Only a prefix is displayed for space. A real comparison uses the complete digest.

A hash needs a trusted reference

If an attacker can replace both a file and its displayed hash, recomputing the hash does not identify the sender.

A digest obtained through a trusted channel can detect a changed download.

A keyed message authentication code, such as HMAC, lets holders of a shared secret check message authenticity and integrity.

The signer uses a private key to sign a message. A verifier uses the corresponding public key to check the message and signature.

Successful verification binds the signed bytes to the signing key under the scheme's assumptions.

Identity depends on trusted key ownership. A signature does not make the content true or confidential.

Creating a digital signature



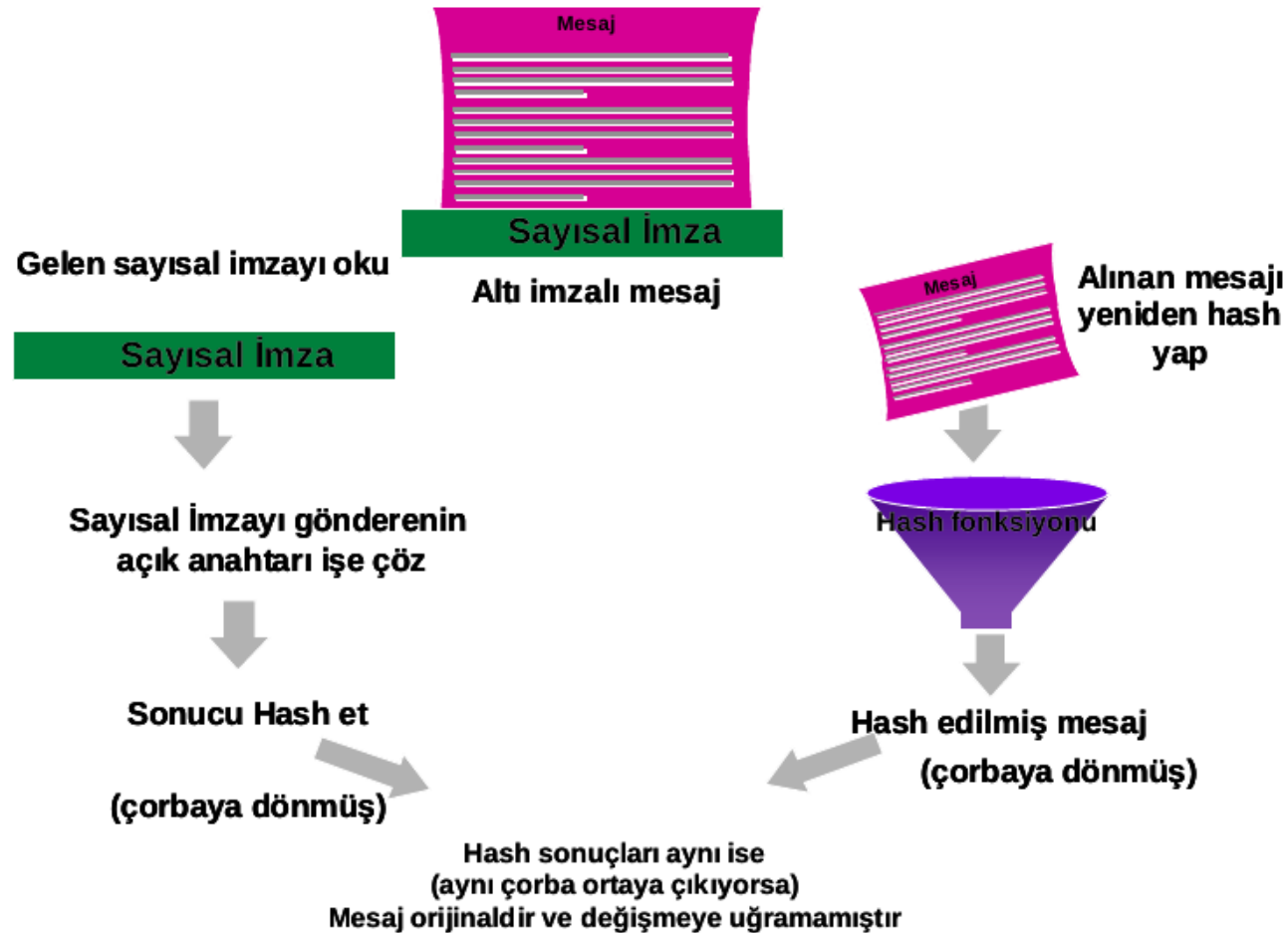
The message influences the signature.

The signer keeps the signing key private.

Use a defined signature scheme, not an improvised hash-and-encrypt construction.

Read “encrypt with the private key” here as a signing operation, not general-purpose encryption.

Verifying a digital signature



Verify against the received message.

An altered message fails verification.

Do not read this as a universal decrypt-and-hash algorithm.

Correction: Verify(public key, message, signature) returns valid or invalid; the left-side extra hash is not a general rule.

Signature verification cases

Received data	Verification with Alice's trusted public key
Original message + valid Alice signature	Pass
Changed message + old signature	Fail
Original message + another key's signature	Fail
Old message replayed with its valid signature	May pass: freshness is a separate issue

A signature authenticates signed bytes; a protocol must also handle replay.

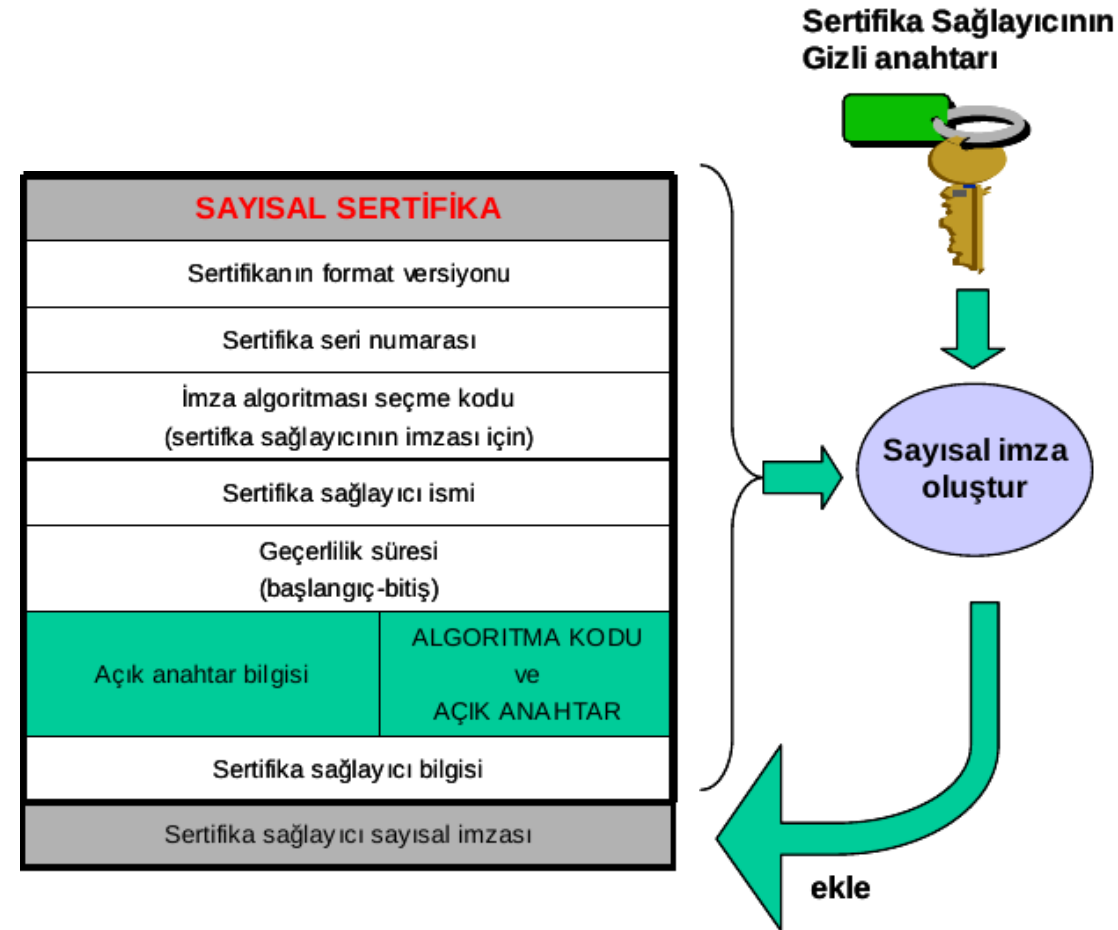
Certificates bind a name to a key

A certificate contains a public key and identity information, signed by an issuer.

A relying party checks a chain to a trusted authority, validity and the intended use.

For a website, it also checks that the requested hostname is covered by the certificate.

Certificate fields and issuer signature



The issuer signs certificate data. The certificate contains the subject's public key, never the subject's private key.

Certificate checks have limits

An expired, untrusted or mismatched certificate should not be treated as proof of the intended website.

Revocation mechanisms address certificates withdrawn before expiry. Checking behavior depends on the application.

A valid website certificate does not establish that the business is honest or the page's claims are true.

HTTPS: a simplified TLS exchange

Stage	Purpose
Client and server negotiate	Select supported protocol parameters
Key agreement and authentication	Establish secrets and verify the server's identity
Protected application traffic	Use symmetric authenticated encryption
Application account login, if needed	Identify the user separately from the server certificate

Conceptual TLS 1.3 overview; the wire protocol includes additional messages and checks.

Where TLS protection ends

TLS protects communication between its endpoints. Data can still be exposed on either endpoint or after storage.

A compromised server can read the plaintext it legitimately receives.

HTTPS does not replace authorization, careful input handling or privacy decisions.

Replay: a valid message at the wrong time

A device receives a previously valid command for a second time. The bytes and signature may still be valid.

A protocol can bind messages to a session and use checked sequence numbers or challenges to establish freshness.

Freshness state must survive or safely reset across reconnects. A sequence number without authentication can itself be forged.

Question	Example
Security: who may access the data?	Only authorized staff can read room history
Privacy: should we collect this data?	Fine-grained occupancy history may reveal routines
Purpose and retention	Keep only what is needed, for an appropriate duration
Disclosure	Share an aggregate rather than identifiable raw history when sufficient

A system can protect stored data well and still collect more information than it should.

Designing a smaller data footprint

Design	What it reveals
Named person + exact timestamp + room	Detailed individual movement
Room + event time	Occupancy patterns may still identify people
Daily count only	Less detail, but possibly enough for planning
No persistent log	Lowest retention, but less evidence for debugging

Choose the least detailed record that serves the stated purpose. Pseudonyms do not guarantee anonymity.

Check: what protection is missing?

A device accepts any correctly formatted command. The connection is encrypted. Can any logged-in user change its settings?

A download and its SHA-256 value are changed together. Does a matching hash prove the download is authentic?

A valid signed command is received twice. Must the second receipt be accepted?

Authorization must restrict who can perform the configuration action. Encryption is a different control.

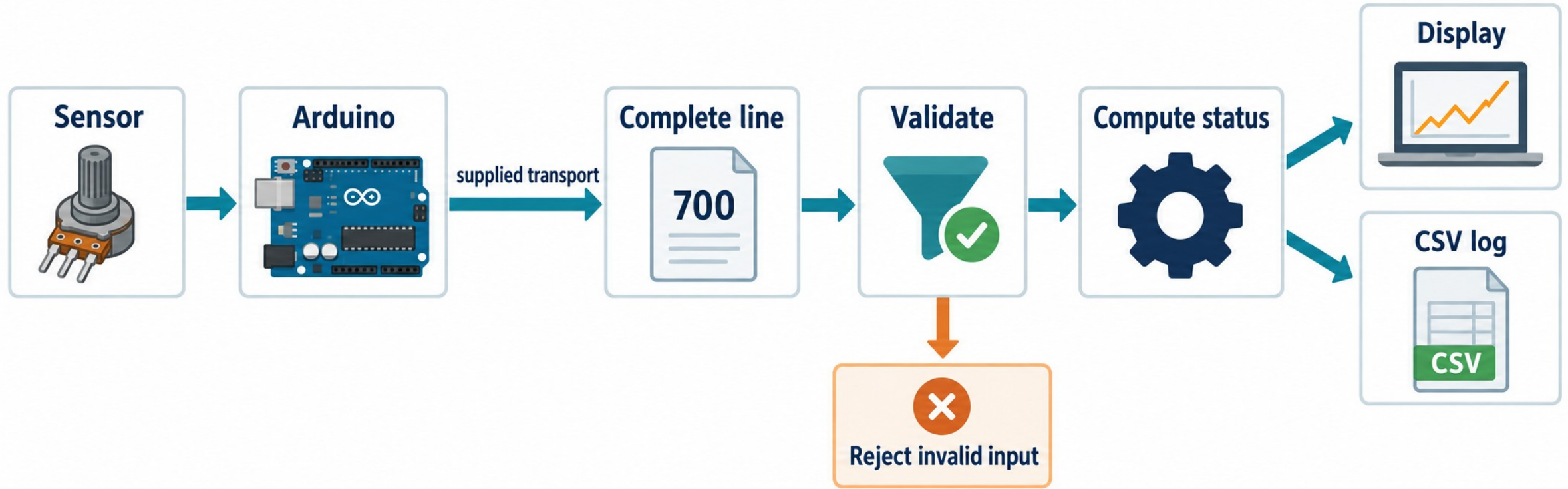
No. The expected hash needs a trusted source, or the file needs an authenticated signature or MAC.

No. The application needs an explicit freshness and duplicate-handling policy.

A complete measurement path

Input contracts, program state and output
Logging and reasoning about failures

Sensor data through the program



The transport helper supplies a complete line. Application code checks and uses it.

A bounded integration example

A familiar Arduino input produces a raw reading. The PC receives one complete line through the existing transport helper.

The PC validates the reading, computes a status, displays the result and writes a small log.

We trace this system using saved input first. No new peripheral or serial library is needed.

Define the input contract

Property	Rule in this example
Content	One unsigned decimal integer
Range	0 through 1023, assuming a 10-bit reading
Framing	One record per line; helper removes CR/LF
Length	1 to 4 digits; no sign, spaces or other characters

The 10-bit range is an explicit example assumption; it is not universal across Arduino boards.

Arduino produces a record

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    int raw = analogRead(A0);  
    Serial.println(raw);  
    delay(200);  
}
```

Arduino C++ example.

Assumes a previously used
analog input and 10-bit
readings.

The PC helper matches the
baud rate and assembles
complete lines.

Bytes arrive before records

A read operation may receive “7”, then “00\r”, then “\n”. It may also receive several lines at once.

The supplied helper buffers bytes until a complete line is available and reports overlong records or a disconnect before the line ends.

The parser below receives a complete, null-terminated string with the line ending removed. It does not parse arbitrary byte fragments.

Which records should be accepted?

Received line	Decision	Reason
0 / 1023	Accept	Endpoints of the allowed range
700	Accept	Valid decimal reading
1024 / -1	Reject	Outside range / forbidden sign
70x / empty / 12345	Reject	Invalid character / no digits / too long

Prediction: does rejecting a record change the last accepted reading?

A parser with a clear result

```
int parse_reading(const char *s, int *out) {  
    if (!s || !out || !*s) return 0;  
    int v = 0;  
    size_t n = 0;  
    for (; s[n]; ++n) {  
        if (n == 4 || s[n] < '0' || s[n] > '9')  
            return 0;  
        v = v * 10 + (s[n] - '0');  
        if (v > 1023) return 0;  
    }  
    *out = v;  
    return 1;  
}
```

Caller supplies a valid C string.

At most four digits: arithmetic stays bounded.

Write through out only after all checks pass.

Trace the parser

Input	Successive values of v	Result
700	$7 \rightarrow 70 \rightarrow 700$	Accept; write 700
1024	$1 \rightarrow 10 \rightarrow 102 \rightarrow 1024$	Reject; out unchanged
70x	$7 \rightarrow 70 \rightarrow \text{invalid character}$	Reject; out unchanged
Empty string	No digit processed	Reject; out unchanged

Failure must not leave a half-parsed value in the application state.

Compute a status without I/O

```
int above_threshold(int raw, int threshold) {  
    return raw >= threshold;  
}
```

```
/* Example configuration */  
const int threshold = 600;
```

```
/* 599 -> 0, 600 -> 1, 601 -> 1 */
```

Use a raw threshold in the same units as the reading.

A pure function is easy to test with saved values.

This is a display status, not a safety control.

Keep related values in one record

```
struct Sample {  
    unsigned number;  
    int raw;  
    int high;  
};  
  
struct Sample sample = {  
    accepted + 1u, raw,  
    above_threshold(raw, 600)  
};
```

Record number counts
accepted samples.

All three fields describe the
same observation.

Structs build on Week 11; no
new data structure is
required.

An end-to-end trace

Line	Accepted number	Raw	Status
599	1	599	0
bad	Unchanged	Unchanged	Reject
600	2	600	1
601	3	601	1

Only accepted readings create sample records. Rejections use a separate error path.

State after a rejected input

Do not replace the last valid value with a partially parsed value or an invented zero.

The old value may be retained, but the interface must indicate that it is old or that a new sample failed.

A status of HIGH describes a valid sample. A communication failure needs its own state.

Write a small CSV record

```
int log_sample(FILE *f,  
               const struct Sample *s) {  
    if (fprintf(f, "%u,%d,%d\n",  
               s->number, s->raw, s->high) < 0)  
        return 0;  
    return fflush(f) == 0;  
}
```

Header: number,raw,high

Check write and flush results.

Flush success does not
guarantee survival of a power
failure.

Output and logging can fail separately

A reading may be valid even when writing the log fails. Do not relabel it as an input error.

Report the logging failure clearly and apply the application's chosen policy. Our test example stops.

Check fclose too. File operations can report delayed errors.

Failure paths to reason about

Event	Required distinction
Disconnected input	No new reading; not a measured zero
Invalid complete line	Reject this record; parser output unchanged
Overlong line / disconnect mid-line	Report framing failure; do not parse a fragment
Log write fails	Valid sample may exist; persistence failed

An error message should identify the failing stage without exposing credentials or unnecessary personal data.

Testing the system in layers

Test the parser with boundary and malformed inputs. Test the status function independently.

Feed saved lines through the combined PC path and compare the complete log with the expected records.

Then use the existing hardware and helper to check framing, connection settings and actual observations.

Input validation and security

Control	What it establishes
Digit and range checks	The input meets this format and value contract
Message authentication	The message was produced by a party holding the relevant key
Authorization	That party may perform the requested operation
Freshness checks	An old accepted message is not silently reused

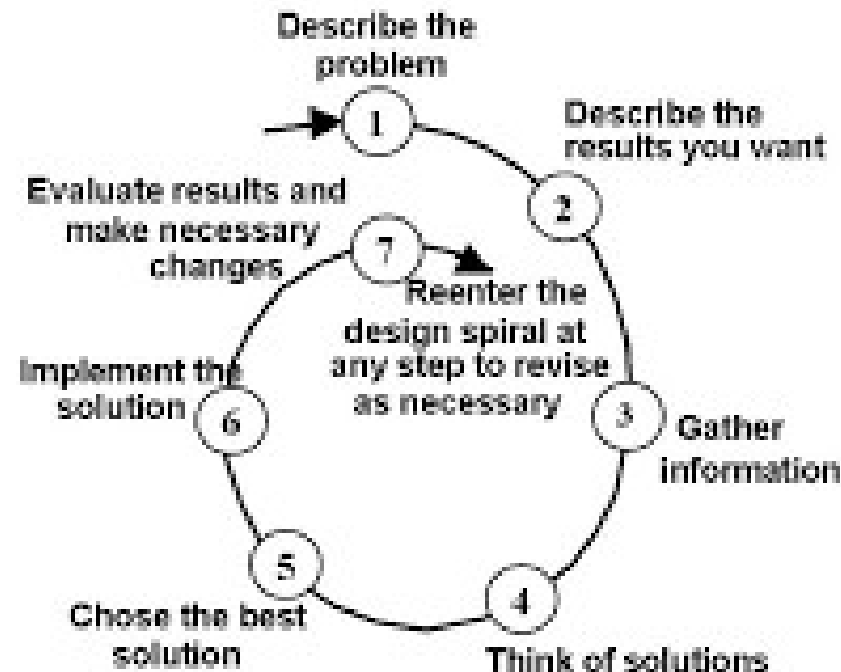
Validation is necessary, but a well-formed reading can still be forged.

For this demonstration, record a sample number, raw value and computed status. Personal identifiers are unnecessary.

For a real deployment, decide who may read the log, how long to retain it and how to detect failures.

Ordinary CSV files are useful evidence for debugging. They are not tamper-proof audit records.

The Technological Method of Problem Solving



Week 7 recap: change one bounded component, observe the result and revise the design.

Calendar Week 13 / Lab 12 consolidates C/Arduino input, computation, output and a small log.

Use the familiar transport helpers and hardware. A bounded modification is enough to exercise the complete path.

The security lesson supports design reasoning; implementing cryptography is not an added lab requirement.

Check: explain the failure

The input is “1024”. Which stage rejects it, and what happens to the current output value?

The input is “700”, but the disk write fails. Is this a parsing error?

Someone sends a forged, correctly formatted “700”. Which checks are insufficient?

The parser rejects the range and leaves its output argument unchanged. The application marks the failed update appropriately.

No. Parsing and computation succeeded; persistence failed and must be reported separately.

Syntax and range checks do not authenticate the sender. A real adversarial setting requires additional protocol protections.

Original figures and further study

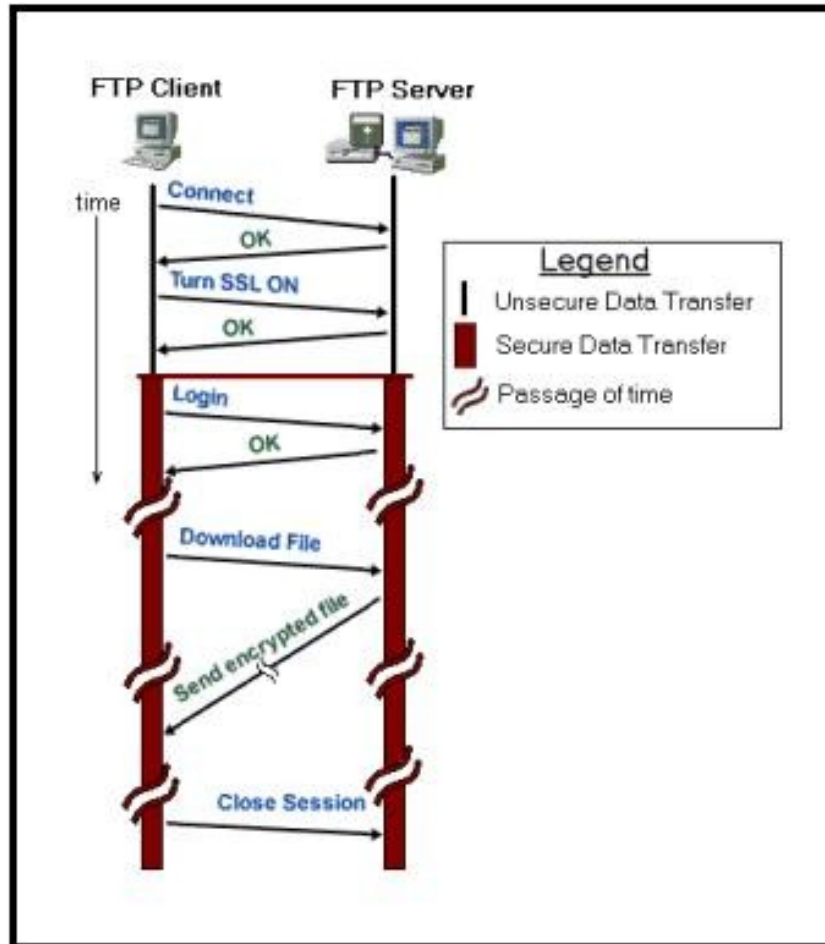
Historical examples with explicit corrections
Primary references for the main concepts

Historical certificate example

Certificate serial number	3082030830820271A003020102020436F2A2E3
Signature algorithm ident for CA	300D06092A864886F70D0101050500
Issuer X.500 name	3009060355040613025553, 3016060355040A130F552E532E20476F7665726E6D656E74, 301C060355040B13154465706172746D656E74206F66204A757374696365
Validity period	301E170D3939303430363139333235365A170D3032303430363230303235365A
Subject X.500 name	3009060355040613025553, 3016060355040A130F552E532E20476F7665726E6D656E74, 301C060355040B13154465706172746D656E74206F66204A757374696365, 30110603550403130A63657274746573746572
Subject X.500 serial number	300A06035504051303303030
Subject public key information	300D06092A864886F70D0101050500
Certificate Extensions	
CRL Distribution Point	30690603551D1F04623060305EA05CA05AA4583056310B30603550406130255 5331183016060355040A130F552E532E20476F7665726E6D656E74311E301C5 5040B13154465706172746D656E74206F66204A75737469610D300B06035504 03130443524C31
Key Usage	300B0603551D0F040403020520
Issuer unique identifier	301F0603551D23041830146BF3A0494A651430A3D08F8274C8DFF40575204A
Subject unique identifier	301D0603551D0E04160414EAB61B64CBA6E9EFA5BA327814D31F06EC5F09
Basic Constraints	30090603551D1304023000
Certificate format version	301906092A864886F67D074100040C300A1B0456342E3003020490
CA Signature	300D06092A864886F70D0101050500

Historical certificate encoding illustration. Field presence varies; do not treat legacy algorithms as current recommendations.

Historical protected-transfer illustration



Protection begins before credentials and file transfer.

The actual protocol has additional checks and messages.

FTP over TLS and SFTP are different protocols.

The figure uses the historical term SSL. Use TLS for contemporary secure transport; this is not a TLS 1.3 message trace.

AES is a symmetric block cipher. SHA-256 is a hash. Neither is a digital-signature scheme.

RSA can be used in separately defined encryption and signature schemes. RC4 is not RSA and is not a signature algorithm.

Use current, reviewed implementations and protocols. The classroom shift cipher is only a way to understand reversibility.

Further study: mechanisms

Brookshear & Brylow: sections 3.5 and 4.5 for security concepts; section 9.7 for data and social implications.

NIST SP 800-175B Rev. 1: distinguish encryption, hashes, MACs, signatures and key establishment.

NIST FIPS 197, FIPS 180-4 and FIPS 186-5: AES, secure hashes and digital signatures.

Further study: identity and systems

NIST SP 800-63B-4: authentication and password-storage guidance.

IETF RFC 5280 and RFC 8446: certificate validation and the TLS 1.3 protocol.

NIST Privacy Framework: consider data-processing risks alongside security risks.