Ideal Functionality in Security and Privacy Analysis

Why Formal Security Definitions?

Informal statements like:

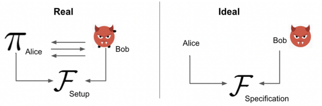
- "Data is encrypted"
- "System preserves privacy"
- "Protocol is secure"

are inadequate because:

- Adversaries are adaptive and computationally powerful
- Systems interact with complex environments
- Security must hold under composition

Goal: Define security as *indistinguishability from an ideal world*.

Two executions:



Real World

Actual protocol Π executed by parties with adversary $\mathcal{A}$.

Ideal World

Trusted party computes a specified functionality $\mathcal{F}$. Adversary replaced by simulator $\mathcal{S}$.

Security holds if no environment $\mathcal{Z}$ can distinguish:

$$\text{Real execution} \approx \text{Ideal execution}$$

What is an Ideal Functionality?

An **ideal functionality** $\mathcal{F}$ is an abstract trusted service:

- Receives inputs from parties
- Computes prescribed output
- Returns outputs to appropriate parties
- Models minimum information leak and enforces privacy and correctness automatically

Key idea:

If the real protocol behaves like $\mathcal{F}$, it is secure.

Example: Secure Message Transmission

Ideal functionality $\mathcal{F}_{\mathsf{SMT}}$:

1. Sender submits message m
2. Functionality delivers m to receiver
3. Adversary learns only allowed leakage (e.g., message length, control flow)

Guarantees:

- Perfect confidentiality (as per definition)
- Perfect integrity (as per definiton)
- Guaranteed delivery (unless model allows blocking)

Any protocol emulating $\mathcal{F}_{\mathsf{SMT}}$ provides secure communication.

Example: Secure Multiparty Computation

Functionality $\mathcal{F}_f$ for computing function f:

1. Parties submit private inputs $x_1, \ldots, x_n$
2. Compute $y = f(x_1, \ldots, x_n)$
3. Return outputs to designated parties

Privacy guarantee:

No party learns anything beyond its input and output.
Do they learn who are the parties?

Captures voting, auctions, statistics, etc.

Simulator and Indistinguishability

Adversary in real world: $\mathcal{A}$
Simulator in ideal world: $\mathcal{S}$

$\mathcal{S}$ must reproduce everything $\mathcal{A}$ sees using only:

- Allowed leakage from $\mathcal{F}$
- Outputs received by corrupted parties

Security condition:

$$\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$$

for all environments $\mathcal{Z}$.

Why Ideal Functionality Matters

Provides:

- Precise specification of security goals
- Composability guarantees
- Modularity in protocol design
- Separation of concerns: WHAT vs HOW

Key insight:
Design protocols to realize ideal services.

Universal Composability (UC)

UC framework (Canetti):

- Protocol secure even when composed with arbitrary others
- Environment can interact concurrently
- Models real-world system complexity

If protocol Π UC-realizes $\mathcal{F}$:

$$\Pi \text{ can safely replace the trusted functionality.}$$

Privacy Through Ideal Functionality

Privacy is encoded as:

- Restricted information flow in $\mathcal{F}$
- Explicit leakage functions
- Corruption models (honest-but-curious, malicious)

Example:
Database query functionality may reveal:

- Query result
- Access pattern
- Nothing else

Limitations and Challenges

- Writing correct functionality is difficult
- Some tasks impossible without setup assumptions
- Efficiency gaps between ideal and real implementations
- Subtle leakage channels may be overlooked

Security proofs depend critically on modeling choices.

Security Goals in Electronic Voting

An e-voting system must satisfy multiple properties simultaneously:

- **Correctness:** Votes are counted accurately
- **Privacy:** Ballots remain secret
- **Eligibility:** Only authorized voters vote
- **Uniqueness:** One vote per voter
- **Verifiability:** Outcome can be independently checked
- **Coercion resistance:** Voters cannot prove how they voted

These goals are formalized via an ideal functionality $\mathcal{F}_{vote}$.

Ideal Voting Functionality $\mathcal{F}_{vote}$

$\mathcal{F}_{vote}$ models a trusted election authority.

Setup:

- Receives list of eligible voters
- Initializes empty ballot box

Voting phase:

1. Voter V_i submits vote v_i
2. Check eligibility and uniqueness
3. Store vote securely

Tally phase:

- Compute result $R = f(v_1, \ldots, v_n)$
- Output R to all parties

Privacy in $\mathcal{F}_{vote}$

Ideal privacy guarantee:

No one learns how any individual voted.

Adversary may learn only:

- Final tally
- Participation information (who voted)
- Allowed leakage (model dependent)

Equivalent to a perfectly secret ballot box.

Verifiability in the Ideal Model

Two key notions:

Individual Verifiability

A voter can confirm their vote was recorded.

Universal Verifiability

Anyone can check that the announced tally is correct.

In $\mathcal{F}_{vote}$, correctness is automatic — no trust in authorities needed.
Real protocols must emulate this property cryptographically.

Coercion Resistance

Strong requirement unique to voting:

A voter cannot prove how they voted — even if bribed or threatened.

Ideal functionality enforces this by:

- Not providing transferable proof of vote
- Allowing voters to produce fake transcripts

This prevents vote buying and coercion.

Adversarial Capabilities

Models typically consider:

- Corrupted voters (static or dynamic)
- Malicious election authorities
- Network attackers
- Coercers or vote buyers

The simulator must reproduce adversary observations using only:

- Allowed leakage
- Public outputs

Real-World Mechanisms Emulating $\mathcal{F}_{vote}$

End-to-end verifiable (E2E-V) protocols approximate the ideal box using:

- Homomorphic encryption
- Mix networks (mixnets)
- Zero-knowledge proofs
- Blind signatures
- Secure multiparty computation

These techniques ensure:

- Ballot secrecy
- Correct tally computation
- Public auditability

Corruption Models

Typical corruption assumptions:

- **Static corruption:** Adversary chooses parties at start
- **Adaptive corruption:** Parties corrupted during execution
- **Honest-but-curious:** Follow protocol but leak state
- **Malicious:** Arbitrary deviations allowed

Voting protocols must tolerate realistic adversarial behavior.

Interfaces and Notation

Parties:

- Voters $V_1, \ldots, V_n$
- Public bulletin board / public output channel (modeled by $\mathcal{F}$ emitting public messages)

Parameters:

- Vote space $\mathcal{V}$ (e.g., candidates, rankings)
- Tally function $T : \mathcal{V}^n \to \mathcal{R}$ (with $\perp$ for abstentions)
- Eligibility set $L \subseteq \{V_1, \ldots, V_n\}$

Adversarial control:

- Corruption set $C \subseteq \{V_1, \ldots, V_n\}$ (static or adaptive)

Explicit Leakage: Minimal and Parameterized

We make leakage explicit via two functions:

$$\mathrm{Leak}_{\mathrm{cast}}(i, v, \mathrm{st}) \quad \text{and} \quad \mathrm{Leak}_{\mathrm{tally}}(\mathbf{v}, \mathrm{st})$$

Minimal (typical) leakage choices:

- Casting leakage: $\mathrm{Leak}_{\mathrm{cast}}(i, v, \mathrm{st}) := i$ (reveals only that V_i cast a ballot)
- Tally leakage: $\mathrm{Leak}_{\mathrm{tally}}(\mathbf{v}, \mathrm{st}) := T(\mathbf{v})$ (reveals only the final outcome)

Optional knobs (if you want to model them):

- ballot length / format class (e.g., for ranked-choice)
- timing / ordering of casts
- total turnout $|\{i : V_i \text{ voted}\}|$

Ideal Voting Functionality $\mathcal{F}_{\mathsf{Vote}}^{\mathsf{leak}}$ (State)

$\mathcal{F}_{\mathsf{Vote}}^{\mathsf{leak}}$ maintains:

- Phase $\phi \in \{\mathsf{open}, \mathsf{closed}\}$
- For each voter i: status $\sigma_i \in \{\mathsf{notcast}, \mathsf{cast}\}$
- Stored ballot vector $\mathbf{v} \in (\mathcal{V} \cup \{\bot\})^n$ initially all $\bot$

Also maintains corruption set C (if adaptive, updated by Corrupt messages).

$\mathcal{F}_{\text{Vote}}^{\text{leak}}$: Casting

Upon receiving (Cast, i, v) from V_i:

1. If $\phi \neq \text{open}$ then ignore.
2. If $V_i \notin L$ then ignore.
3. If $\sigma_i = \text{cast}$ then ignore (or allow overwrite if your model permits re-voting).
4. Set $\mathbf{v}[i] \leftarrow v$ and $\sigma_i \leftarrow \text{cast}$.
5. Compute leakage $\ell \leftarrow \text{Leak}_{\text{cast}}(i, v, \text{st})$.
6. Send ℓ to adversary $\mathcal{A}$ (and/or post publicly, depending on model).

Note: The vote value v is *never* leaked for honest voters unless allowed by the leakage function.

$\mathcal{F}_{\text{Vote}}^{\text{leak}}$: Tally and Output

Upon receiving (Close) from an authorized trigger (e.g., a public clock or trustees): set $\phi \leftarrow$ closed.

Upon receiving (Tally):

1. If $\phi \neq$ closed then ignore (or close automatically).
2. Compute $R \leftarrow T(\mathbf{v})$.
3. Compute tally leakage $\lambda \leftarrow \text{Leak}_{\text{tally}}(\mathbf{v}, \text{st})$.
4. Output (Result, λ) publicly to all parties.
5. Additionally, reveal to $\mathcal{A}$ the votes of corrupted voters:

$$\{(i, \mathbf{v}[i]) : V_i \in C\}.$$

Minimal leakage: public output is exactly (Result, R).

Modeling Active Corruption

If adaptive corruption is modeled:
Upon $(\text{Corrupt}, i)$ from $\mathcal{A}$:

- Add i to C.
- Return internal state for V_i to $\mathcal{A}$ (including $\mathbf{v}[i]$ if already cast).

This makes explicit what an adaptive adversary learns when it corrupts a voter mid-election.

Coercion Model and Interface

We extend $\mathcal{F}_{\text{Vote}}^{\text{leak}}$ to $\mathcal{F}_{\text{CRV}}^{\text{leak}}$ by modeling coercion sessions.

Adversary capabilities:

- May designate voters as coerced
- May demand specific ballots
- Receives a view of the voting interaction

Key goal:

A coerced honest voter can cast any vote while producing a transcript consistent with adversarial demands.

Functionality maintains a coercion set $K \subseteq L$.

$\mathcal{F}_{\mathsf{CRV}}^{\mathsf{leak}}$: Coercion Control

Upon $(\mathsf{Coerce}, i, \hat{v})$ from $\mathcal{A}$:

1. Add i to coercion set K
2. Record demanded vote $\hat{v}_i \leftarrow \hat{v}$
3. Notify voter V_i that it is under coercion

Interpretation:

- $\hat{v}_i$ is the vote the coercer expects
- Honest voter may choose a different true vote

$\mathcal{F}_{\text{CRV}}^{\text{leak}}$: Casting Under Coercion

Upon (Cast, i, v) from V_i:

1. Process eligibility and uniqueness as in $\mathcal{F}_{\text{Vote}}^{\text{leak}}$
2. Store **true vote** $\mathbf{v}[i] \leftarrow v$
3. If $i \notin K$:
   - Leak $\text{Leak}_{\text{cast}}(i, v, \text{st})$ to $\mathcal{A}$
4. If $i \in K$ (coerced voter):
   - Provide adversary with simulated vote

$$\tilde{v}_i \leftarrow \hat{v}_i$$

   - Send $(\text{FakeCast}, i, \tilde{v}_i)$ to $\mathcal{A}$

True vote remains hidden unless voter is corrupted.

Tally and Privacy for Coerced Voters

During tally:

- Result computed from true vote vector $\mathbf{v}$
- Fake votes $\tilde{v}_i$ are ignored

Adversary additionally learns:

$$\{(i, \mathbf{v}[i]) : i \in C\} \quad \text{(corrupted voters only)}$$

Thus for honest coerced voters:

Adversary cannot determine whether the voter obeyed.

Security Intuition: Simulation View

Let V_i be an honest coerced voter.
Adversary's view consists of:

- Fake ballot $\tilde{v}_i$
- Allowed leakage from Leak
- Final tally

Coercion resistance requires that this view is simulatable without knowing the true vote v_i.

Equivalently:

Compliance and defiance are indistinguishable.

Real and Ideal Executions

Real world:

- Parties run protocol Π (the actual e-voting scheme)
- Adversary $\mathcal{A}$ controls corrupted parties and network scheduling
- Environment $\mathcal{Z}$ provides inputs and observes outputs

Ideal world:

- Parties interact only with $\mathcal{F}_{\mathsf{Vote}}^{\mathsf{leak}}$
- Adversary replaced by simulator $\mathcal{S}$

Exact UC-Style Indistinguishability

Let $\mathsf{EXEC}^{\mathsf{real}}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa)$ be the output bit of $\mathcal{Z}$ when interacting with Π and $\mathcal{A}$ at security parameter κ.
Let $\mathsf{EXEC}^{\mathsf{ideal}}_{\mathcal{F}^{\mathsf{leak}}_{\mathsf{Vote}},\mathcal{S},\mathcal{Z}}(\kappa)$ be the output bit of $\mathcal{Z}$ when interacting with $\mathcal{F}^{\mathsf{leak}}_{\mathsf{Vote}}$ and $\mathcal{S}$.

Definition (UC realization)

Π UC-realizes $\mathcal{F}^{\mathsf{leak}}_{\mathsf{Vote}}$ if:

$$\forall\,\mathsf{PPT}\ \mathcal{A}\ \exists\,\mathsf{PPT}\ \mathcal{S}\ \forall\,\mathsf{PPT}\ \mathcal{Z}: \quad \mathsf{EXEC}^{\mathsf{real}}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa) \approx \mathsf{EXEC}^{\mathsf{ideal}}_{\mathcal{F}^{\mathsf{leak}}_{\mathsf{Vote}},\mathcal{S},\mathcal{Z}}(\kappa)$$

where $\approx$ denotes computational indistinguishability of ensembles.

Equivalently:

$$\left|\Pr[\mathsf{EXEC}^{\mathsf{real}} = 1] - \Pr[\mathsf{EXEC}^{\mathsf{ideal}} = 1]\right| \leq \mathsf{negl}(\kappa).$$

Interpreting “Minimum Leakage” via Leak

The leakage functions *parameterize* what privacy means.

Ballot secrecy beyond the tally: for honest voters, the ideal world exposes nothing about $\mathbf{v}[i]$ except what is inferable from:

$$\mathsf{Leak}_{\mathsf{cast}} \;+\; \mathsf{Leak}_{\mathsf{tally}} \;+\; \{(i, \mathbf{v}[i]) : i \in C\}.$$

Thus, to claim “privacy”, you must commit to specific Leak:

- participation-only leakage, or
- participation+timing, or
- turnout only, etc.