An Approach to Formalize Information-theoretic Security of Multiparty Computation Protocols Across Cryptographic Paradigms

Cheng-Hui Weng[1], Reynald Affeldt[2], Jacques Garrigue[3], Takafumi Saikawa[3]

[1]Nagoya University and Mercari Inc. weng.cheng.hui.c4@math.nagoya-u.ac.jp

[2]National Institute of Advanced Industrial Science and Technology

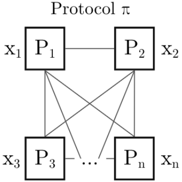
[3]Nagoya University

JSIAM Annual Meeting September 2025

Background: Secure-Multiparty Computation

Secure-multiparty computation (hereafter, SMC) refers to a *n* parties cryptographic protocol that implements an *n*-ary function *F* securely:

$$\mathrm{F}(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{n}) = \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n}$$

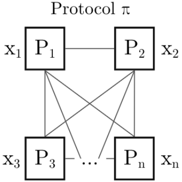


$$\rightsquigarrow \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n}$$

Background: Secure-Multiparty Computation

Secure-multiparty computation (hereafter, SMC) refers to a *n* parties cryptographic protocol that implements an *n*-ary function *F* securely:

$$\mathrm{F}(\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{n}) = \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n}$$



$\rightsquigarrow$ $\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n}$

Example: The SMC Scalar Product Protocol (SMC-SPP) [Du and Zhan(2002)]

P_a and P_b each has a private vector, and they want to compute $F = \cdot$ (scalar product) collaboratively :

$$P_a : \overrightarrow{x_a} \to y_a \qquad P_b : \overrightarrow{x_b} \to y_b$$

where

$x_a \cdot x_b = \overrightarrow{y_a} + \overrightarrow{y_b}$
P_a cannot guess x_b, and
P_b cannot guess x_a

The Security Guarantees of SMC Protocols

SMC protocols are described in the literature in various ways:

1. **Pseudocode and natural language**: when describing the protocol
2. **Mathematical language**: when providing proofs for security
3. **Programming language**: when implementing the protocol

We want to provide a better integration to improve security.

Outline

1. **Our Approach**: Formalization using Interpretation
2. **Correctness using Interpretation**
3. **Privacy using Interpretation**
4. **Use-case 1**: SMC Scalar Product Protocol (SMC-SPP)
5. **Use-case 2**: Distributed and Secure Dot-Product (SMC-DSDP)
6. **Conclusion and Future Work**

Our Tool: ROCQ prover

ROCQ: Programming language / Interactive theorem prover

Our Tool: Rocq prover

Rocq: Programming language / Interactive theorem prover

One can write:

- Programs
- Specification for the programs
- Mathematical proof of the specification

Our Tool: Rocq prover

Rocq: Programming language / Interactive theorem prover

One can write:

- Programs
- Specification for the programs
- Mathematical proof of the specification

And the Rocq typechecker assures that the proof contains no mistake.

Our Approach: Formalization using Interpretation

In ROCQ, we introduce a sublanguage of π-calculus to describe SMC protocols as programs:

Our Approach: Formalization using Interpretation

In ROCQ, we introduce a sublanguage of π-calculus to describe SMC protocols as programs:

```
Inductive proc : Type :=
        | Init of data & proc         (* Register input to trace *)
        | Send of ℕ & data & proc     (* Send to nth process *)
        | Recv of ℕ & (data → proc)   (* Receive from nth process *)
        | Ret  of data                (* Return result *)
        | Finish                      (* Finish successfully *)
        | Fail.                       (* Finish with failure *)
```

Each protocol party is represented by one process, in type `proc`.

We later define a corresponding interpreter for execution and verification.

Interpretation according to Rewriting Rules

As in the π-calculus, in our language, computation can be described through rules rewriting a configuration:

$$
\begin{aligned}
i : \texttt{Init}\ x.\ p \ \mid\ C &\xrightarrow{\ i \leftarrow x\ } i : p \ \mid\ C \\
i : \texttt{Send}\ j\ x.\ p \ \mid\ j : \texttt{Recv}\ i\ f. \ \mid\ C &\xrightarrow{\ j \leftarrow x\ } i : p \ \mid\ j : f\ x \ \mid\ C \\
i : \texttt{Ret}\ x \ \mid\ C &\xrightarrow{\ i \leftarrow x\ } i : \texttt{Finish} \ \mid\ C
\end{aligned}
$$

$i \leftarrow x$ denotes that the interpreter recording *process* i *receives the value* x

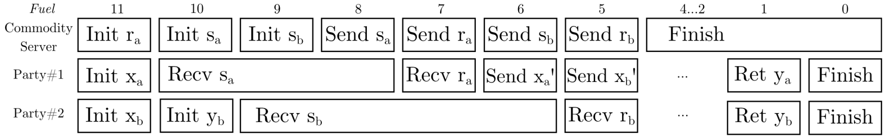
Interpretation according to Rewriting Rules

As in the π-calculus, in our language, computation can be described through rules rewriting a configuration:

$$
\begin{array}{rcl}
i : \texttt{Init}\ x.\ p \mid C & \xrightarrow{i \leftarrow x} & i : p \mid C \\
i : \texttt{Send}\ j\ x.\ p \mid j : \texttt{Recv}\ i\ f. \mid C & \xrightarrow{j \leftarrow x} & i : p \mid j : f\ x \mid C \\
i : \texttt{Ret}\ x \mid C & \xrightarrow{i \leftarrow x} & i : \texttt{Finish} \mid C
\end{array}
$$

$i \leftarrow x$ denotes that the interpreter recording *process i receives the value x*

The actual interpretation of the SMC-SPP program is:



Interpretation according to Rewriting Rules

The final *input traces* of each SMC-SPP process are:

$$
\begin{aligned}
P_c &: (r_a, \overrightarrow{s_b}, \overrightarrow{s_a}) \\
P_a &: (y_a, t_a, \overrightarrow{x_b'}, r_a, \overrightarrow{s_a}, \overrightarrow{x_a}) \\
P_b &: (y_b, \overrightarrow{x_a'}, r_b, \overrightarrow{s_b}, y_b, \overrightarrow{x_b})
\end{aligned}
$$

Proofs using Interpretation

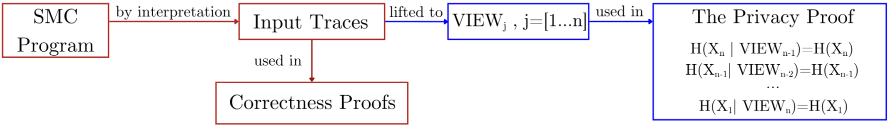
Based on input traces, we verify the following protocol properties:

1. Correctness of the protocol
2. Privacy of secret inputs

Proofs using Interpretation

Based on input traces, we verify the following protocol properties:

1. Correctness of the protocol
2. Privacy of secret inputs

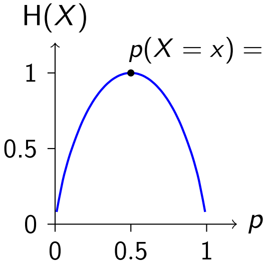


Recap: Shannon Entropy

The **Shannon entropy** $\mathrm{H}(X)$ measures the uncertainty of a discrete random variable X with probability mass function $p(x)$:

$$\mathrm{H}(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

- Measures the average uncertainty (information content) of X.
- Maximum when all outcomes are equally likely.
- Minimum (zero) when X is deterministic.



Joint Random Variable, Joint Entropy and Conditional Entropy

Intuitively:

1. Joint random variable $\langle X, Y \rangle$ is valued in X- and Y-axes.
2. Joint entropy $H(\langle X, Y \rangle)$ is the amount of knowledge about $\langle X, Y \rangle$.
3. Conditional entropy $H(X \mid Y)$ is the amount of knowledge about X after knowing Y.

Relation: $H(X \mid Y) = H(\langle X, Y \rangle) - H(Y)$

Joint Random Variable, Joint Entropy and Conditional Entropy

Intuitively:

1. Joint random variable $\langle X, Y \rangle$ is valued in X- and Y-axes.
2. Joint entropy $H(\langle X, Y \rangle)$ is the amount of knowledge about $\langle X, Y \rangle$.
3. Conditional entropy $H(X \mid Y)$ is the amount of knowledge about X after knowing Y.

Relation: $H(X \mid Y) = H(\langle X, Y \rangle) - H(Y)$

If X and Y are independent:

$$H(\langle X, Y \rangle) = H(X) + H(Y)$$
$$H(X \mid Y) = H(X)$$

(Knowing Y does not increase the knowledge of X)

Definition: Privacy

We follow the pen-and-paper proof work [Shen et al.(2007)] to define the privacy property of SMC protocols using *conditional entropy*:

$$H(X_i \mid VIEW_j^{\pi}) = H(X_i)$$

Meaning:

1. After the protocol π,
2. The knowledge that party j gains (denoted by $VIEW$),
3. About party i's secret X_i,
4. Is equal to what party j knows before the protocol execution.

In other words: party j cannot gain any new knowledge about X_i by executing the protocol.

Use-case 1: SMC Scalar Product Protocol (SMC-SPP)

The SMC-SPP is the first use-case of our language:

The SMC Scalar Product Protocol (SMC-SPP) [Du and Zhan(2002)]

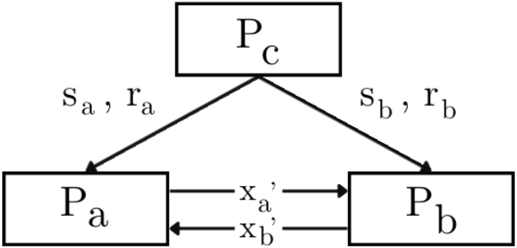
P_a, P_b each has a private vector, and
they compute $F = \cdot$ (scalar product) collaboratively :

$$\begin{array}{l} P_a : \overrightarrow{x_a} \to y_a \\ P_b : \overrightarrow{x_b} \to y_b \end{array} \quad \textit{where} \quad \begin{array}{l} \overrightarrow{x_a} \cdot \overrightarrow{x_b} = y_a + y_b \\ P_a \text{ cannot guess } x_b \\ P_b \text{ cannot guess } x_a \end{array}$$

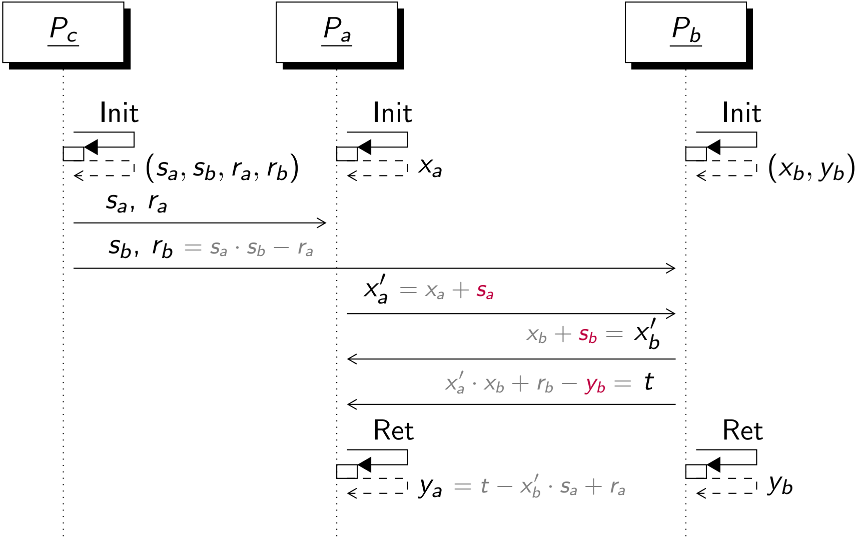
SMC-SPP was used in a real-world public health research [Chen et al.(2012)].

SMC-SPP: the Role of Each Party

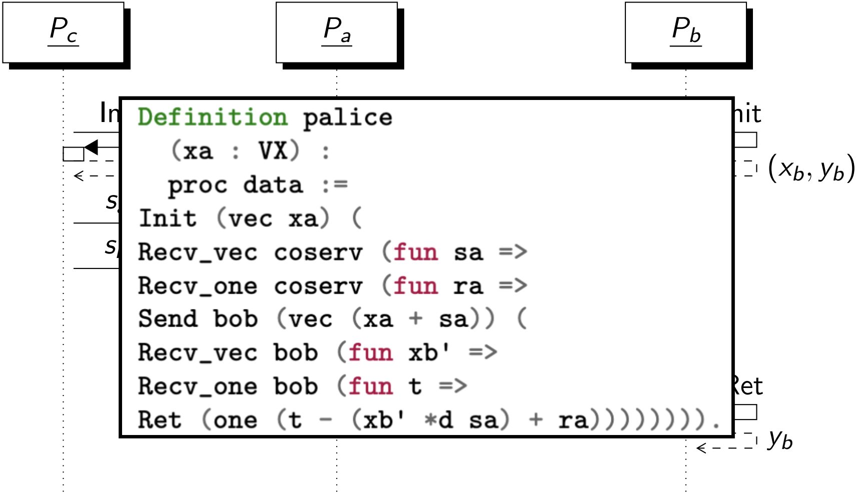
The protocol has the 3rd party P_c (commodity server).
It only issues random values for the two parties:



Sequence Diagram of SMC-SPP



SMC-SPP Implementation Example: the P_a Program




```
Definition palice
  (xa : VX) :
  proc data :=
Init (vec xa) (
Recv_vec coserv (fun sa =>
Recv_one coserv (fun ra =>
Send bob (vec (xa + sa)) (
Recv_vec bob (fun xb' =>
Recv_one bob (fun t =>
Ret (one (t - (xb' *d sa) + ra)))))))).
```

Proof of Correctness using Interpretation

Intermediate variables in input traces preserve how they are computed:

$$P_c : (r_a, \overrightarrow{s_b}, \overrightarrow{s_a})$$
$$P_a : (y_a = t_a - \overrightarrow{x_b'} \cdot \overrightarrow{s_a} + r_a,\ t_a = \overrightarrow{x_a'} \cdot \overrightarrow{x_b} + r_b - y_b,\ \overrightarrow{x_b'} = \overrightarrow{x_b} + \overrightarrow{s_b},\ r_a, \overrightarrow{s_a}, \overrightarrow{x_a})$$
$$P_b : (y_b, \overrightarrow{x_a'} = \overrightarrow{x_a} + \overrightarrow{s_a},\ r_b = \overrightarrow{s_a} \cdot \overrightarrow{s_b} - r_a,\ \overrightarrow{s_b}, y_b, \overrightarrow{x_b})$$

Proof of Correctness using Interpretation

Intermediate variables in input traces preserve how they are computed:

$$P_c : (r_a, \overrightarrow{s_b}, \overrightarrow{s_a})$$
$$P_a : (y_a = t_a - \overrightarrow{x_b'} \cdot \overrightarrow{s_a} + r_a,\ t_a = \overrightarrow{x_a'} \cdot \overrightarrow{x_b} + r_b - y_b,\ \overrightarrow{x_b'} = \overrightarrow{x_b} + \overrightarrow{s_b},\ r_a, \overrightarrow{s_a}, \overrightarrow{x_a})$$
$$P_b : (y_b, \overrightarrow{x_a'} = \overrightarrow{x_a} + \overrightarrow{s_a},\ r_b = \overrightarrow{s_a} \cdot \overrightarrow{s_b} - r_a,\ \overrightarrow{s_b}, y_b, \overrightarrow{x_b})$$

Therefore, we can prove the correctness of the SMC-SPP results mostly done by the automatic ROCQ tactic `ring` in a few lines.

Theorem 1 (SMC Scalar Product)

Let $\overrightarrow{x_a}$, $\overrightarrow{x_b}$, y_a and y_b be corresponding variables from `smc_scalar_product_traces`, then $\overrightarrow{x_a} \cdot \overrightarrow{x_b} = y_a + y_b$.

Because:

$$\overrightarrow{x_a} \cdot \overrightarrow{x_b} = \overbrace{\overrightarrow{x_a} \cdot \overrightarrow{x_b} + \overrightarrow{s_a} \cdot \overrightarrow{x_b} + (\overrightarrow{s_a} \cdot \overrightarrow{s_b} - r_a) - y_b - (\overrightarrow{s_a} \cdot \overrightarrow{x_b} + \overrightarrow{s_a} \cdot \overrightarrow{s_b}) + r_a}^{y_a} + y_b$$

Privacy Proof using Interpretation

We formalize the pen-and-paper proofs of the privacy-preserving property of SMC-SPP as follows:

Theorem 2 (SMC-SPP Preserves Privacy)

Let $(Y_1, T_1, \overrightarrow{X_2'}, R_1, \overrightarrow{S_1}, \overrightarrow{X_1})$ *and* $(Y_2, \overrightarrow{X_1'}, R_2, \overrightarrow{S_2}, Y_2, \overrightarrow{X_2})$ *be party views of* P_a *and* P_b *from the lifted* `smc_scalar_product_traces`*, then*

$$H(\overrightarrow{X_1} \mid Y_2, \overrightarrow{X_1'}, R_2, \overrightarrow{S_2}, Y_2, \overrightarrow{X_2}) = H(\overrightarrow{X_1}) \text{ and}$$
$$H(\overrightarrow{X_2} \mid Y_1, T_1, \overrightarrow{X_2'}, R_1, \overrightarrow{S_1}, \overrightarrow{X_1}) = H(\overrightarrow{X_2}).$$

Privacy Proof using Interpretation

We formalize the pen-and-paper proofs of the privacy-preserving property of SMC-SPP as follows:

Theorem 2 (SMC-SPP Preserves Privacy)

Let $(Y_1, T_1, \overrightarrow{X_2'}, R_1, \overrightarrow{S_1}, \overrightarrow{X_1})$ *and* $(Y_2, \overrightarrow{X_1'}, R_2, \overrightarrow{S_2}, Y_2, \overrightarrow{X_2})$ *be party views of* P_a *and* P_b *from the lifted* `smc_scalar_product_traces`*, then*

$$H(\overrightarrow{X_1} \mid Y_2, \overrightarrow{X_1'}, R_2, \overrightarrow{S_2}, Y_2, \overrightarrow{X_2}) = H(\overrightarrow{X_1}) \text{ and}$$
$$H(\overrightarrow{X_2} \mid Y_1, T_1, \overrightarrow{X_2'}, R_1, \overrightarrow{S_1}, \overrightarrow{X_1}) = H(\overrightarrow{X_2}).$$

Key: we need to *lift* the input traces to information-theoretic *party views*

Lifting Input Traces to Party Views

By applying the function to inputs we get input traces.

deterministic function and inputs *input traces*

$$\boxed{\text{SMC Protocol Program}}\ \left(\mathrm{x_a},\ \mathrm{x_b},\ \mathrm{r_a},\ \mathrm{s_a},\ \mathrm{s_b}\right) = ((\mathrm{r_a},\ \mathrm{s_b},\ \mathrm{s_a}),\ (\mathrm{y}_a,\ \mathrm{t}_a,\ \mathrm{x'}_\mathrm{b},\ \mathrm{r}_a,\ \mathrm{s_a},\ \mathrm{x_a}),\ (\mathrm{y}_b,\ \mathrm{x'}_\mathrm{a},\ \mathrm{r}_b,\ \mathrm{s_b},\ \mathrm{y}_b,\ \mathrm{x_b}))$$

Lifting Input Traces to Party Views

By applying the function to inputs we get input traces.

deterministic function and inputs *input traces*

$$\boxed{\text{SMC Protocol Program}} \left(\mathrm{x_a},\ \mathrm{x_b},\ \mathrm{r_a},\ \mathrm{s_a},\ \mathrm{s_b}\right) = ((\mathrm{r_a},\ \mathrm{s_b},\ \mathrm{s_a}),\ (\mathrm{y}_a,\ \mathrm{t}_a,\ \mathrm{x'_b},\ \mathrm{r}_a,\ \mathrm{s_a},\ \mathrm{x_a}),\ (\mathrm{y}_b,\ \mathrm{x'_a},\ \mathrm{r}_b,\ \mathrm{s_b},\ \mathrm{y}_b,\ \mathrm{x_b}))$$

The program is lifted by composing random variables of inputs.

deterministic function and random variables *party views of random variables*

$$\boxed{\text{SMC Protocol Program}} \circ \langle \mathrm{X_1},\ \mathrm{X_2},\ \mathrm{R_1},\ \mathrm{S_1},\ \mathrm{S_2} \rangle = \langle\ \langle \mathrm{R_1},\ \mathrm{S_2},\ \mathrm{S_1} \rangle,\ \langle \mathrm{Y_1},\ \mathrm{T_1},\ \mathrm{X'_2},\ \mathrm{R_1},\ \mathrm{S_1},\ \mathrm{X_1} \rangle,\ \langle \mathrm{Y_2},\ \mathrm{X'_1},\ \mathrm{R_2},\ \mathrm{S_2},\ \mathrm{Y_2},\ \mathrm{X_2} \rangle\ \rangle$$

where $\langle f, g \rangle(x) = (f(x), g(x))$

Use-case 2: Distributed and Secure Dot-Product (SMC-DSDP)

We are currently formalizing this protocol using our interpretation-based approach.

1. SMC-DSDP [Dumas et al.(2017)] is an N-party ($N > 2$) protocol.
2. It utilizes homomorphic cryptographic systems.
3. The original work provides a security proof for the three-party case.

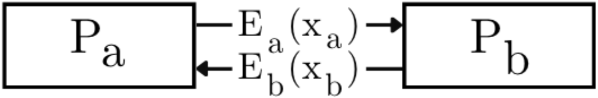
The Distributed and Secure Dot-Product Protocol (SMC-DSDP) [Dumas et al.(2017)]

P_a, P_b, and P_c they compute $S = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3$ collaboratively:

$$\begin{array}{lcl} P_a : (u_1, u_2, u_3, v_1, r_2, r_3) \rightarrow S & & P_a \text{ cannot guess } v_2 \text{ and } v_3 \\ P_b : v_2 \rightarrow \beta & \textit{where} & P_b \text{ cannot guess } v_1 \text{ and } v_3 \\ P_c : v_3 \rightarrow \gamma & & P_c \text{ cannot guess } v_1 \text{ and } v_2 \end{array}$$

The major difference between SMC-SPP and SMC-DSDP is how secrets are masked:

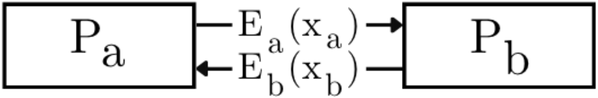
1. SMC-SPP: adding with uniformly distributed random values
2. SMC-DSDP: encrypting by public keys



Security using Homomorphic Cryptographic Systems

The major difference between SMC-SPP and SMC-DSDP is how secrets are masked:

1. SMC-SPP: adding with uniformly distributed random values
2. SMC-DSDP: encrypting by public keys



→ The receiver can *add* or *multiply* over the encrypted secrets without decrypting them.

Progress

1. We extended our method to prove SMC-DSDP's correctness
2. We also analyized its securety using the conditional entropy
3. We found SMC-DSDP leaks information *but* is still safe

Progress

1. We extended our method to prove SMC-DSDP's correctness
2. We also analyized its securety using the conditional entropy
3. We found SMC-DSDP leaks information *but* is still safe

The authors distinguish *safety* from *security*:

1. Security: parties learn only their inputs and the protocol output
2. Safety: the protocol output does not reveal secret inputs

$\rightarrow$ we integrate both by our method.

SMC-DSDP Security via Simulation-based Verification

Case P_a**:** The adversary corrupts P_a, who learns the final result $S = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3$.

Security means: the adversary still cannot learn more than its own inputs and the protocol output.

SMC-DSDP Simulation-based Verification

Let

$$View_{P_a} = (U, R, \gamma, S, A, B, C)$$

denote P_a's view of messages and γ, A, B, C are encrypted, and let

$$View_{Sim_a} = (U, R, \gamma', S, A', B', C')$$

where A', B', and C' are simulated and γ' is inferred from S.

If the adversary can distinguish $View_{P_a}$ from $View_{Sim_a}$, it will break the IND-CPA assumption of the cryptosystem (which is assumed impossible).
$\rightarrow$ P_a cannot learn more than its own inputs and the output. □

Conditional Entropy Reveals Leakage

For SMC-DSDP, privacy for V_2 requires:

$$H(V_2 \,|\, View_{P_a}) = H(V_2)$$

Conditional Entropy Reveals Leakage

For SMC-DSDP, privacy for V_2 requires:

$$H(V_2 \mid View_{P_a}) = H(V_2)$$

But trying to prove it will disclose that this only holds if

$$(V_2, V_3) \cdot (U_2, U_3) \perp\!\!\!\perp V_2$$

which is generally false.

$\rightarrow$ the protocol leaks information about V_2.

Conditional Entropy Reveals Leakage

For SMC-DSDP, privacy for V_2 requires:

$$H(V_2 \,|\, View_{P_a}) = H(V_2)$$

But trying to prove it will disclose that this only holds if

$$(V_2, V_3) \cdot (U_2, U_3) \perp\!\!\!\perp V_2$$

which is generally false.

$\rightarrow$ the protocol leaks information about V_2.

Indeed: knowing $(V_2, V_3) \cdot (U_2, U_3)$ lets P_a restrict possible (V_2, V_3) values, since P_a knows (U_2, U_3) and the domain is finite.

Conditional Entropy Reveals Leakage

For SMC-DSDP, privacy for V_2 requires:

$$H(V_2 \mid View_{P_a}) = H(V_2)$$

But trying to prove it will disclose that this only holds if

$$(V_2, V_3) \cdot (U_2, U_3) \perp\!\!\!\perp V_2$$

which is generally false.

$\rightarrow$ the protocol leaks information about V_2.

Indeed: knowing $(V_2, V_3) \cdot (U_2, U_3)$ lets P_a restrict possible (V_2, V_3) values, since P_a knows (U_2, U_3) and the domain is finite.

$\rightarrow$ However, the protocol is still *safe* even with this leakage.

Formalizing Harmless Leakage in Security Proofs

1. **Conditional entropy**
   $H(V_2 \mid View_{P_a}) < H(V_2)$ quantifies the leakage.
2. **Subset restriction**
   V_2 is restricted to the set $\{V_2 \mid (V_2, V_3) \cdot (U_2, U_3) = S\}$ for observed S, but not uniquely determined.
3. **Harmless leakage**
   By the Rouché–Capelli theorem, if unknown variables (V_2, V_3) are more than the number of linearn independent equations $((V_2, V_3) \cdot (U_2, U_3) = S)$, the system has multiple solutions; thus, the adversary cannot reconstruct the secret.

$\rightarrow$ We are formalizing this harmless leakage by combining learn algebra and information theory.

Conclusion and Future Work

An *interpretation based methodology* combining the following ideas:

1. Protocol description language
2. Process calculus
3. Execution evidence from the interpretation (input traces)
4. Information theory

All in the same framework: from the protocol to all proofs.

Metrics: when we completed the formalization of SMC-SPP, we had:

1. Just 40 lines of code for defining the interpreter (reusable)
2. 80 lemmas and theorems (57 of them are reusable)
3. 574 lines of proof in total

(We are currently refactoring the code, so these metrics may change.)

Future work:

1. Complete the formalization of combining linear algebra and information theory, and complete the formalization of SMC-DSDP.

Kung Chen, Tsan-sheng Hsu, Wen-Kai Huang, Churn-Jung Liau, and Da-Wei Wang. 2012.
Towards a Scripting Language for Automating Secure Multiparty Computation. In *First Asia-Pacific Programming Languages and Compilers Workshop (APPLC 2012), Beijing, China, June 14, 2012*.

Wenliang Du and Justin Zhijun Zhan. 2002.
A practical approach to solve Secure Multi-party Computation problems. In *Workshop on New Security Paradigms (NSPW 2002), Virginia Beach, VA, USA, September 23-26, 2002*. ACM, 127–135.
doi:10.1145/844102.844125

Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. 2017.
Dual protocols for private multi-party matrix multiplication and trust computations.
Comput. Secur. 71 (2017), 51–70.
doi:10.1016/J.COSE.2017.04.013

Chih-Hao Shen, Justin Zhan, Da-Wei Wang, Tsan-Sheng Hsu, and Churn-Jung Liau. 2007.
Information-Theoretically Secure Number-Product Protocol. In *2007 International Conference on Machine Learning and Cybernetics*, Vol. 5. 3006–3011.
doi:10.1109/ICMLC.2007.4370663