Skja: Adversarial Logic on Steroids

AKA SAI LALITH KUMAR, University of Colorado Boulder, USA

We introduce *Skja*, a new incorrectness logic [16] which extends on *Adversarial Logic* [19] with spi-calculus. Adversarial logic is very interesting in that it lets you apply under-approximation for exploitability analysis. The current core limitation of adversarial logic is that it only allows for parallel composition over multiple programs but it has problems with the model. With Skja, we bring the rich theory of spi-calculus to let the logic be composable for any infinite arbitrary set of programs which may be adversarial or not. This lets us encode and talk about attacks like Man-in-the-middle (MiTM) attacks and DDoS (Denial of Service) attacks. For this end, we first prove that parallel composition is a monoid over the adversarial logic grammar. We change the denotational semantics and program rules to support n-ary parallel composition while also proving that the changes do not effect the soundness. We also show a reasoning of Skja on Needham-Schroeder to highlight

1 INTRODUCTION

When one reasons about programmes, we have access to a rich variety of logics, and one of them would be incorrectness logic [16] which is equivalent to [9]. The logic targets under-approximating the post of a condition c; this is symmetrical to the work done by Hoare[11] which over-approximates. The way we can understand this through diagrammatic representation is by visualising the state space as a hierarchy of set inclusions, taken from O'Hearn's presentation. As shown in Figure 1, the central arrow represents the *strongest postcondition* ($post(c)$), which describes exactly the set of states reachable by the program.

$$\begin{tikzcd}
 & & \textit{Predicates} & \text{(Hoare Logic)} \\
\textit{Predicates} \arrow[rru, "\{P\}c\{Q\}"] \arrow[rr, "post(c)\text{ (Exact)}"] \arrow[rrd, "{[P]c[Q]}"'] & & \textit{Predicates} \arrow[u, "\subseteq"'] & \\
 & & \textit{Predicates} \arrow[u, "\subseteq"'] & \text{(Incorrectness Logic)}
\end{tikzcd}$$

Fig. 1. The hierarchy of program logic. Hoare logic (top) over-approximates the reachable states, while Incorrectness logic (bottom) under-approximates them.

In this hierarchy, the top layer corresponds to standard Hoare Logic. It provides an *over-approximation*, meaning the postcondition Q must contain all reachable states ($post(c) \subseteq Q$). This is useful for proving safety (i.e., "nothing bad happens").

Conversely, the bottom layer represents under-approximation. Here, the postcondition Q must be a *subset* of the reachable states ($Q \subseteq post(c)$). This logic is useful for proving the existence of bugs or specific reachable states (i.e., "something bad definitely happens"). Now, you can also observe, the triangle for top is rather short compared to the triangle for the bottom. This is a delibrate choice to highlight the fact that over-approximation enjoyed 5 decades of being in the front seat [1–3] and hence the closeness to the perfect postcondition.

Author's address: Aka Sai Lalith Kumar, sailalithkumar.aka@colorado.edu, University of Colorado Boulder, USA.

To showcase an example with under-approximation, we take a rather simple looking but tricky example derived from [9] below comparing the validity of the triplets shown below:

$$\langle T \rangle \quad x := y;\ z := x \quad \langle z = y \rangle \qquad \text{Not Valid}$$

This triplet is invalid in under-approximation logic. While it is true that $z = y$ in the final state, the assertion $\langle z = y \rangle$ as a set includes states where $x \neq y$ (e.g., state $\{x = 0, z = 1, y = 1\}$). However, the program guarantees $x = y$. Because the assertion includes "extra" states that the program logic cannot reach, it is not a valid subset (under-approximation) of the true postcondition.

$$\langle T \rangle \quad x := y;\ z := x \quad \langle z = y \wedge x = y \rangle \qquad \text{Valid}$$

This triplet is valid. By conjoining $x = y$, we restrict the postcondition to exclude the unreachable states. The resulting set matches exactly the behavior of the code, satisfying the requirement $Q \subseteq post(c)$. Application of under-approximation to security reasoning is a very novel yet fundamental contribution [19]. And this contribution brings proving software or protocols are secure to the realm of finding bugs or vulnerabilities aka exploitability analysis [10].

In the context of security auditing, the cost of a "false positive" is exceptionally high. Static analysis tools that over-approximate often drown developers in warnings that may never manifest in reality [18]. This "boy who cried wolf" phenomenon desensitizes engineers to security alerts. Incorrectness logic flips this paradigm: if the logic derives a state where a secret is leaked, that leak is guaranteed to be reachable. There are no false alarms, only confirmed kill chains. This property is invaluable for exploit generation, where the goal is to produce a concrete attack trace rather than an abstract guarantee of safety.

Contributions. In this paper, we bridge the gap between incorrectness logic and process calculi. Specifically:

- We formalize parallel composition as a commutative monoid over the adversarial logic grammar, proving that the system can be treated as an unstructured "soup" of processes rather than a rigid hierarchy.
- We introduce *Skja*, extending the semantics of Adversarial Logic with primitives from the spi-calculus to support n-ary multi-party composition.
- We validate our logic by formally deriving the attack trace for the Needham-Schroeder Public Key protocol, demonstrating how Skja naturally captures Man-in-the-Middle scenarios that previous binary models struggled to express.

2 OVERVIEW: WHY ADVERSARIAL LOGIC ISN'T ENOUGH

Before introducing our semantics, we must address why standard Adversarial Logic (AL) [19] is insufficient for modeling complex protocols, necessitating an upgrade to the spi-calculus model. We identify three core friction points:

1. The Monolithic State Problem. Standard AL models the world as a rigid couple (σ_p, σ_a)—the program and the adversary. This forces a separation of concern that is unnatural for multi-party protocols. In a scenario like Needham-Schroeder, we have Alice, Bob, and potentially Charlie (the attacker). AL struggles to distinguish between the honest state of Bob and the honest state of Alice; they are lumped into σ_p. Skja dissolves this monolith by mapping Process IDs to local stores ($\Pi(i)$), allowing us to reason about the state of specific agents independently.

2. Fixed Topologies. In AL, the network topology is generally static; the "Program" communicates with the "Adversary" over a pre-defined boundary. However, modern protocols utilize scope extrusion and channel mobility. A server might create a temporary session channel and pass it to a client, dynamically altering the network topology. The spi-calculus treats channels as first-class citizens that can be encrypted and sent as messages. Skja adopts this power, allowing us to reason about attacks where the structure of the network changes during the attack—such as an adversary injecting themselves into a session by stealing a session ID (channel) that didn't exist when the protocol started.

3. Replication and Asynchrony. Finally, modeling infinite behavior via the replication operator ($!P$) is native to spi-calculus but cumbersome in standard AL. While our current implementation focuses on finite traces for exploit generation, building on the spi-calculus foundation prepares Skja for future extensions involving unbounded replication, such as modeling Distributed Denial of Service (DDoS) attacks where N attackers flood a target.

3 NEW SEMANTICS

In this section we build all the semantics and rules which transform adversarial logic to Skja. First, we assume the reader is familiar with the semantics of adversarial logic and refer them to the original paper for the full set [19], our work states what is new to the established semantics.

First we change the state space. In AL, states were rigid couples (σ_p, σ_a). To support n-ary composition, we map Process IDs to their local stores.

Definition 1 (General State Space). $\Sigma ::= [PIDs \rightharpoonup Variables \rightarrow Values]$

Let $\Pi \in \Sigma$ be a global state. We denote $\Pi(i)$ as the local store of process i. We also maintain the global channel state $\Gamma \in [Channels \rightarrow List(Values)]$. A configuration is a tuple $\langle \Pi, \Gamma \rangle$.

A key notation change from adversarial logic is that instead of using $[\epsilon : P]$ where $\epsilon \in \{ad, ok\}$, we define Δ as a mapping from PIDs to assertions.

Definition 2 (Generalized Triple). $[\Delta_{pre}]C[\Delta_{post}]$

For processes Alice (A), Bob (B), and Eve (E), Δ maps identities to assertions: $\{A : P_A, B : P_B, E : P_{Eve}\}$. The triple $[\Delta]C[\Delta']$ is valid if, for all processes $i \in dom(\Delta)$, the local execution satisfies the under-approximate relation between $\Delta(i)$ and $\Delta'(i)$.

3.1 Syntax of Assertions

To reason about what an adversary knows, we define the syntax for expressions and assertions, adapting the model from [19].

Definition 3 (Syntax).

$$\begin{aligned} Variables\ V &::= x \mid n \mid \alpha \\ Expressions\ E &::= V \mid rand() \mid enc(E, K) \mid \langle E, E \rangle \\ Assertions\ P, Q &::= E = E \mid Knows(E) \mid P \wedge Q \mid \exists x.P \end{aligned}$$

Here, Knows(E) is a predicate indicating that the value E is derivable from the agent's current knowledge set using Dolev-Yao deduction rules.

3.2 Semantics

Definition 4 (Small-step semantics for Composition). *Let $C_1 \parallel \cdots \parallel C_n$ be a composition of n processes.*

- ***Independent Step:*** *If process i takes a local step $\langle \sigma_i, \Gamma \rangle \rightarrow \langle \sigma_i', \Gamma \rangle$, then:*

$$\langle \Pi[i \mapsto \sigma_i], \Gamma \rangle \rightarrow \langle \Pi[i \mapsto \sigma_i'], \Gamma \rangle$$

- ***Communication Step:*** *If process i writes to channel c and process j reads from channel c, they synchronize:*
$$\langle \Pi, \Gamma \rangle \xrightarrow{i!c(v), j?c(v)} \langle \Pi', \Gamma' \rangle$$
where Π' updates the local variables of i and j appropriately.
- ***Termination:*** *$skip \parallel \cdots \parallel skip$ is the terminal configuration.*

We restate the properties of channels and local variables for processes from adversarial logic.

Hypothesis 1 (Ownership Principle).

- *No two parallel processes share the same local variables and are distinctly separated (Disjointness). $\forall i \neq j, dom(\Pi(i)) \cap dom(\Pi(j)) = \emptyset$.*
- *They can only communicate through channels, which we state are global and represented by Γ.*

3.3 Inference Rules

To support the n-ary composition defined above, we extend the rules of adversarial logic by modifying the

1. The n-Par Rule. This rule allows us to combine independent proofs of n processes. It relies on the disjointness hypothesis.

$$\textsc{n-Par}\quad \frac{\forall i \in \{1..n\}, \quad [\Delta_{pre}(i)]C_i[\Delta_{post}(i)]}{[\Delta_{pre}]C_1 \parallel \cdots \parallel C_n[\Delta_{post}]}$$

2. The n-Com Rule. This is the workhorse of protocol verification when it comes to Skja. It synchronizes a sender i and a receiver j over channel c. Other processes k remain idle (frame property).

$$\textsc{n-Com}\quad \frac{[\Delta_{pre}(i)]\mathrm{out}(c,v)[\Delta_{post}(i)] \qquad [\Delta_{pre}(j)]\mathrm{in}(c,x)[\Delta_{post}(j)] \qquad \forall k \notin \{i,j\}, \Delta_{pre}(k) = \Delta_{post}(k)}{[\Delta_{pre}]\ (\cdots \parallel \mathrm{out}_i \parallel \cdots \parallel \mathrm{in}_j \parallel \dots)\ [\Delta_{post}]}$$

Semantically, this implies a logical flow of the value v from i to j:

$$\exists v, (\Delta_{pre}(i) \implies x = v) \wedge (\Delta_{post}(j) \implies x = v)$$

3.4 The Dolev-Yao Adversary Model

To reason about protocols, we adopt the standard Dolev-Yao model, assuming the network is under total control of the adversary. The adversary can read, block, and modify messages but cannot break cryptography without the correct keys.

We formalize the attacker's capabilities using the deduction relation $\vdash$. Let $\mathcal{K}$ be the set of messages currently known to the attacker. The relation $\mathcal{K} \vdash m$ (read: "from knowledge $\mathcal{K}$, message m can be derived") is defined inductively:

$$\frac{m \in \mathcal{K}}{\mathcal{K} \vdash m}(\text{Axiom}) \quad \frac{\mathcal{K} \vdash m_1 \quad \mathcal{K} \vdash m_2}{\mathcal{K} \vdash \langle m_1, m_2 \rangle}(\text{Pairing}) \quad \frac{\mathcal{K} \vdash \langle m_1, m_2 \rangle}{\mathcal{K} \vdash m_i}(\text{Proj})$$

$$\frac{\mathcal{K} \vdash m \quad \mathcal{K} \vdash k}{\mathcal{K} \vdash \{m\}_k}(\text{Encrypt}) \qquad \frac{\mathcal{K} \vdash \{m\}_k \quad \mathcal{K} \vdash k^{-1}}{\mathcal{K} \vdash m}(\text{Decrypt})$$

In Skja, this deduction relation connects the operational semantics to the assertion logic. We define the semantics of the assertion Knows(m) as follows:

$$\sigma \models \text{Knows}(m) \iff \sigma(\text{knowledge}) \vdash m$$

This definition is crucial for under-approximation. Unlike safety verification, where one must compute the potentially infinite set of all derivable knowledge to prove a secret is *never* leaked, Skja only requires demonstrating the existence of a single derivation tree. If there exists a sequence of rule applications allowing the adversary to construct the message m needed for the next step of the attack trace, the assertion holds. This makes the check constructive and efficient for exploit generation.

4 PROOFS

4.1 Proof of Composition (||) forms a monoid

The reason we would like to have composition as a commutative monoid is that it frees us from thinking about ordering and composition of operators, allowing us to treat the system as a "soup" of processes, consistent with the chemical abstract machine model often used for process calculi.

First, we define the structural congruence relation (≡) for our logic. This relation identifies process terms that are syntactically different but semantically identical.

Definition 5 (Structural Congruence). *The parallel composition operator* $\parallel$ *satisfies the following laws:*

$$P \parallel Q \equiv Q \parallel P \qquad \textit{(Commutativity)} \qquad (1)$$

$$(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \qquad \textit{(Associativity)} \qquad (2)$$

$$P \parallel \mathit{skip} \equiv P \qquad \textit{(Identity)} \qquad (3)$$

Theorem 1. *(Process, ||, skip) form a commutative monoid.*

Proof: A commutative monoid has three properties for it to satisfy. We show that (||) satisfies all below:

- **Commutativity:** For some processes P and Q, $P \parallel Q \equiv Q \parallel P$. Every derivation of a step from $P \parallel Q$ is obtained by applying one of the small-step rules. Swapping the syntactic positions of P and Q gives a derivation of the same step from $Q \parallel P$ (process IDs are intrinsic to the state, not the syntactic position). Therefore the sets of possible traces are identical.
- **Associativity:** For some processes P, Q and R, $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$. The transition relation is defined on the set of PIDs. Grouping $(P \parallel Q)$ is merely syntactic sugar for a set of processes $\{P, Q\}$. The execution semantics operate on the union of disjoint heaps. Since set union is associative, the composition is associative.
- **Identity:** For some process P, $(P \parallel \mathit{skip}) \equiv P$. The *skip* process has no transitions and modifies no state. Its interleaving with P adds no new traces and restricts no existing traces of P.

Lemma 1 (Permutations of $\parallel$). *For any set of processes* $\{C_1, ..., C_n\}$*, any syntactic permutation of their composition results in an equivalent system under the defined denotational semantics.*

4.2 Foundational Lemmas

We prove the soundness of Skja by demonstrating that our inference rules preserve the under-approximation property defined in Adversarial Logic. To do so, we first adapt the Characterization and Substitution lemmas from Vanegue [19] to our n-ary state space Σ.

Lemma 2 (Characterization). *A triple $[\Delta_{pre}]C[\Delta_{post}]$ is valid if and only if every state satisfying the post-condition is reachable from a state satisfying the pre-condition via the operational semantics of C. Formally:*

$$\forall \Pi' \in [\![\Delta_{post}]\!], \exists \Pi \in [\![\Delta_{pre}]\!] \textit{ such that } \langle \Pi, \Gamma \rangle \longrightarrow^* \langle \Pi', \Gamma' \rangle$$

Lemma 3 (Substitution). *For any assertion P, variable x, and value v:*

$$\sigma \in [\![P[v/x]]\!] \iff \sigma[x \mapsto v] \in [\![P]\!]$$

This lemma ensures that asserting a property about a variable x in a post-state is equivalent to asserting the property about the value v assigned to x in the pre-state.

4.3 Soundness of Inference Rules

Theorem 2 (Soundness of n-Par). *The n-Par rule is valid.*

Proof. We must show that if the premises $\forall i, [\Delta_{pre}(i)]C_i[\Delta_{post}(i)]$ hold, then $[\Delta_{pre}]C_1 \parallel \cdots \parallel C_n[\Delta_{post}]$ satisfies the Characterization Lemma.

Let $\Pi' \in [\![\Delta_{post}]\!]$ be a global state satisfying the post-condition. By the definition of our generalized state space, Π' decomposes into disjoint local stores: $\Pi' = \bigcup_i \sigma'_i$ where $\sigma'_i \in [\![\Delta_{post}(i)]\!]$.

By applying the **Characterization Lemma** to each premise individually:

$$\forall i, \exists \sigma_i \in [\![\Delta_{pre}(i)]\!] \text{ such that } \langle \sigma_i, \Gamma \rangle \longrightarrow^* \langle \sigma'_i, \Gamma'_i \rangle$$

We construct the global start state $\Pi = \bigcup_i \sigma_i$. By the **Independent Step** semantics defined in Section 3.2:

$$\text{If } \langle \sigma_i, \Gamma \rangle \longrightarrow \langle \sigma'_i, \Gamma \rangle \text{ then } \langle \Pi[i \mapsto \sigma_i], \Gamma \rangle \longrightarrow \langle \Pi[i \mapsto \sigma'_i], \Gamma \rangle$$

Because the domains of σ_i are disjoint (Hypothesis 1), the local transitions of each process C_i can be interleaved without interference. Therefore, the sequence of independent steps transforms Π into Π'. Since $\Pi \in [\![\Delta_{pre}]\!]$ and $\Pi' \in [\![\Delta_{post}]\!]$, the characterization holds. □

Theorem 3 (Soundness of n-Com). *The n-Com rule is valid.*

Proof. We analyze the synchronization between a sender i and receiver j on channel c. Let $\Pi' \in [\![\Delta_{post}]\!]$. We focus on the receiver's state σ'_j (the sender's side follows similarly).

The premise for the receiver is $[\Delta_{pre}(j)]\text{in}(c, x)[\Delta_{post}(j)]$. The operational semantics for communication state that:

$$\langle \Pi, \Gamma \rangle \xrightarrow{i!c(v), j?c(v)} \langle \Pi', \Gamma' \rangle$$

where the receiver's local store updates as $\sigma'_j = \sigma_j[x \mapsto v]$.

By the premise, we know $\sigma'_j \in [\![\Delta_{post}(j)]\!]$. Substituting the operational update into the assertion, we require $\sigma_j[x \mapsto v] \in [\![\Delta_{post}(j)]\!]$.

We now invoke the **Substitution Lemma** (3). This implies:

$$\sigma_j \in [\![\Delta_{post}(j)[v/x]]\!]$$

The assertion $\Delta_{post}(j)$ implies $x = v$. Therefore, there exists a pre-state σ_j capable of receiving v such that the post-state condition is met. Since the operational semantics explicitly perform the assignment $x \leftarrow v$ that satisfies the logic, and the Characterization Lemma is satisfied for the step, the rule is sound. □

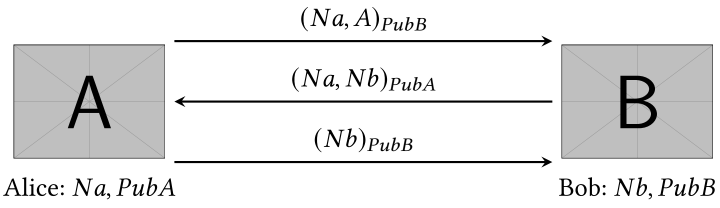

Fig. 2. Needham-Schroeder

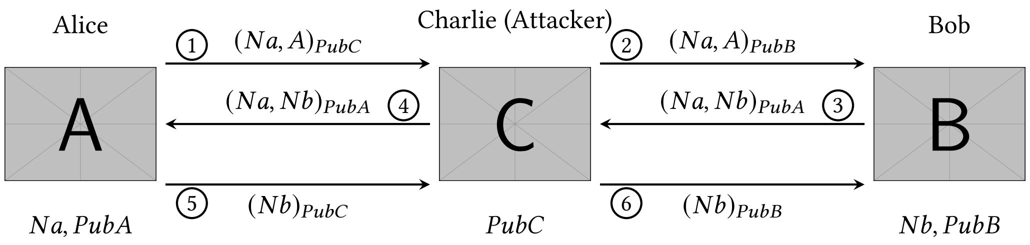

Fig. 3. Lowe's man-in-the-middle attack, where Charlie impersonates Alice.

5 REASONING WITH SJKA: NEEDHAM-SCHROEDER

5.1 Needham-Schroeder

Needham-Schroeder(NS)[15], is an authentication protocol. The way it runs is as follows:

We refer to figure 1, which shows the steps of NS protocol. First we see Alice initiating the protocol with Bob by sending here Nonce Na with her own ID paired together. This is done to make sure Bob knows who he is communicating with. Now, we see Bob responds to Alice with his own Nonce Nb and Na paired together; this is done so that Alice can confirm her nonce and ensure she is not talking to another third party, as only Bob can deduce her nonce. Now, finally, Alice sends back Bob's nonce in the same spirit as above.

This protocol is very sensible but hides a cunning attack in plain sight. We refer the reader to figure 2 for Lowe's attack. Here you see, Charlie is an impersonator. Who can be called a Dolev-Yao adversary as the semantics and powers of the same are defined in the section 2. To recap, he can replay messages, stop messages and forward them and strip messages if they're encrypted with his private key. Now in Lowe's attack, the security guarantees of Alice are maintained, meaning that they won't share the nonce with some other person who they don't know they are talking to, but for Bob it's broken. This can be observed at the end of the protocol run where Bob thinks he is talking to Alice, but it's actually Charlie who now has Bob's and Alice's nonce, so Charlie can act as Alice and keep the conversation going. This is a failure of authentication.

5.2 Formalizing the Attack in Skja

We demonstrate the validity of Lowe's attack by tracing the evolution of the generalized assertion map Δ through the application of the **n-Com** rule.

Let the initial state be Δ_0. The participants have the following initial knowledge:

$$\Delta_0 = \{A : \text{Knows}(Na, PrivA),\ B : \text{Knows}(Nb, PrivB),\ C : \text{Knows}(PrivC)\}$$

Phase 1: Initiation and Impersonation (Steps 1 & 2)

Alice intends to talk to Charlie. She executes $\text{out}(c, \{Na, A\}_{PubC})$. Charlie reads from c. Applying

n-Com:

$$[\Delta_0(A)]\text{out} \cdots \parallel \text{in}[\Delta_1(C)]$$

The synchronization ensures the value flows to Charlie. Since Charlie possesses *PrivC*, he decrypts the message.

$$\Delta_1(C) \implies \text{Knows}(Na)$$

Using this new knowledge, Charlie impersonates Alice. He constructs $\{Na, A\}_{PubB}$ and sends it to Bob. Applying **n-Com** between Charlie and Bob:

$$\Delta_2(B) \implies \text{Received}(\{Na, A\}_{PubB})$$

Bob decrypts this. Crucially, Bob's internal state now asserts he is communicating with Alice.

Phase 2: The Oracle Step (Steps 3 & 4)

Bob responds to the perceived initiator. He sends $\{Na, Nb\}_{PubA}$. Charlie intercepts this via **n-Com**.

$$\Delta_3(C) \implies \text{Knows}(\{Na, Nb\}_{PubA})$$

At this stage, Charlie *cannot* decrypt the message to learn Nb because he lacks *PrivA*. This is where the logic highlights the vulnerability. Charlie forwards the blob opaque to Alice (Step 4). Applying **n-Com** between Charlie and Alice:

$$\Delta_4(A) \implies \text{Received}(\{Na, Nb\}_{PubA})$$

Alice, believing this is a valid response from Charlie (her intended partner), decrypts it. She verifies Na and learns Nb.

Phase 3: The Leak (Step 5)

This is the fatal step. Following the protocol, Alice confirms the session by sending Nb encrypted with her partner's key (*PubC*).

$$[\Delta_4(A)]\text{out}(c, \{Nb\}_{PubC}) \parallel \text{in}(c, x)[\Delta_5(C)]$$

Applying **n-Com**, Charlie receives $x = \{Nb\}_{PubC}$. Since $\Delta_5(C)$ includes *PrivC*:

$$\Delta_5(C) \implies \text{Knows}(Nb)$$

The adversary has successfully extracted the secret nonce Nb.

Phase 4: Completion (Step 6)

Charlie re-encrypts Nb with *PubB* and sends it to Bob via **n-Com**. Bob verifies Nb and completes the session.

Conclusion

The final state Δ_{final} contains a contradiction to the authentication specification:

$$\text{(1)}\quad \Delta_{final}(B) \implies \text{Partner} = Alice$$
$$\text{(2)}\quad \Delta_{final}(C) \implies \text{Knows}(Na, Nb)$$

In Skja, the reachability of Δ_{final} where the adversary knows Nb constitutes a proof of the exploit. The proof script could be more integrated to look like the reasoning of adversarial logic, but we choose to keep it focused on the rules we have defined.

6 RELATED WORK

Related work in program analysis and formal verification of protocols and software comes with a rather rich history. We stand on the shoulders of giants, yet the integration of incorrectness logic with process calculi remains under-explored.

Program Logics. The foundation of our work rests on Incorrectness Logic [16], which inverted the reasoning of standard Hoare Logic [11]. While standard logic proves the absence of bugs, incorrectness logic proves their presence. This is particularly relevant for security, where demonstrating an exploit path is often more valuable than a theoretical safety guarantee.

Process Calculi and Security. The use of process calculi [14] to model security protocols was pioneered by Abadi and Gordon with the spi-calculus This introduced cryptographic primitives directly into the operational semantics of the π-calculus. While automated tools like ProVerif [4], WALDO [12] and Tamarin[13] have successfully applied these concepts using model checking, Skja aims to bring this expressivity into a deductive logic framework.

Abstract Interpretation. Abstract Interpretation [6] is the archetypal framework for static analysis, traditionally used to compute over-approximations of program behaviors to prove safety (i.e., that bad states are unreachable). In the context of Skja and Adversarial Logic, we utilize the framework of Abstract Interpretation but invert the lattice direction. Instead of the lattice of properties over-approximating the set of traces, we operate on the *under-approximate* domain.

As noted by O'Hearn [16] and Vanegue [19], incorrectness logic can be viewed as an abstract interpretation where the domain $\mathcal{D}$ represents subsets of reachable states. For Skja, our domain is enriched with the Dolev-Yao adversary model. We can define our abstract domain $\mathcal{D}^\sharp$ as the product of the program state lattice and the adversary knowledge lattice:

$$\mathcal{D}^\sharp = \mathcal{P}(\Sigma_{prog}) \times \mathcal{K}_{adv}$$

where $\mathcal{K}_{adv}$ represents the set of derivable messages. The abstraction function α maps concrete execution traces to these logical assertions. Our fixpoint computation, therefore, does not find the "smallest set containing all behaviors" (Least Fixpoint in safety analysis), but rather attempts to find a "non-empty set of confirmed behaviors" (identifying satisfying traces for exploitation). This aligns Skja with *under-approximate abstract interpretation*, a growing field that formalizes bug finding as a rigorous static analysis discipline rather than heuristics. There is also rather interesting working showing the connection between abstract interpretation and program logics which state the connection between them [7]. **Adversarial Logic.** The most direct predecessor to Skja is Adversarial Logic [19]. While groundbreaking in applying under-approximation to security, it is limited by its "rigid couple" state space (Program vs. Adversary). Our work generalizes this to an n-ary composition, effectively unblocking the modeling of multi-party protocols and complex topologies that were previously awkward to represent in the original Adversarial Logic framework.

7 CONCLUSION AND FUTURE WORK

In this paper, we adapted under-approximation logic for security and boosted its versatility by integrating techniques from the spi-calculus. This in effect turned the adversarial logic from a rigid monolith to a dynamic system. We demonstrated that this extension preserves the soundness of the underlying logic through formal proofs. Finally, we applied Skja to the Needham-Schroeder protocol, formally proving the validity of the classic Man-in-the-Middle vulnerability. The expressiblity of Skja shines when it comes to multi-parties, especially in the context of IOT and distributed systems. One strong fit we can see is to use Skja to reason about Denial of Service Attacks (DDoS). The spi-calculi based !P operator would be the right fit. An interesting line of future work is to turn Skja into an algorithm which not only verifies but synthesizes the attack traces. This could be achieved through a bounded model checking based search algorithms. Both under-approximation and over-approximation could be combined like in [20] to get benefit of both worlds. We believe this to be an interesting research direction. One could also Certify incorrectness logic and Skja with a dependent type theory based theorem prover like Lean [8], ROCQ [5] and Isabelle[17]. Finally, as reviewed in related work, application of abstract interpretation to underapproximation is exiciting direction.

ACKNOWLEDGMENTS

I thank my advisor, Dr. Gowtham Kaki. I acknowledge Dr. Bor-Yuh Evan Chang and Kirby Linvill for teaching me all I know about PL. Thanks to Dakota Brayan for feedback, and Dr. Fabio Somenzi for teaching me about logic.

REFERENCES

[1] Nahid A Ali. 2017. A survey of verification tools based on hoare logic. *International Journal of Software Engineering & Applications* 8 (2017), 87–100.
[2] Krzysztof R Apt. 1981. Ten years of Hoare's logic: A survey—part I. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 3, 4 (1981), 431–483.
[3] Krzysztof R Apt. 1983. Ten years of Hoare's Logic: a survey—Part II: nondeterminism. *Theoretical Computer Science* 28, 1-2 (1983), 83–109.
[4] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2018. ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. *Version from* 16 (2018), 05–16.
[5] Adam Chlipala. 2013. *Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant.* MIT Press.
[6] Patrick Cousot. 1996. Abstract interpretation. *ACM Computing Surveys (CSUR)* 28, 2 (1996), 324–328.
[7] Patrick Cousot. 2024. Calculational design of [in] correctness transformational program logics by abstract interpretation. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 175–208.
[8] Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. 2015. The Lean theorem prover (system description). In *International Conference on Automated Deduction.* Springer, 378–388.
[9] Edsko De Vries and Vasileios Koutavas. 2011. Reverse hoare logic. In *International Conference on Software Engineering and Formal Methods.* Springer, 155–171.
[10] Sarah Elder, Md Rayhanur Rahman, Gage Fringer, Kunal Kapoor, and Laurie Williams. 2024. A survey on software vulnerability exploitability assessment. *Comput. Surveys* 56, 8 (2024), 1–41.
[11] Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580.
[12] Kirby Linvill, Gowtham Kaki, and Eric Wustrow. 2023. Verifying indistinguishability of privacy-preserving protocols. *Proceedings of the ACM on Programming Languages* 7, OOPSLA2 (2023), 1442–1469.
[13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN prover for the symbolic analysis of security protocols. In *International conference on computer aided verification.* Springer, 696–701.
[14] Robin Milner. 1980. *A calculus of communicating systems.* Springer.
[15] Roger M Needham and Michael D Schroeder. 1978. Using encryption for authentication in large networks of computers. *Commun. ACM* 21, 12 (1978), 993–999.
[16] Peter W O'Hearn. 2019. Incorrectness logic. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–32.
[17] Lawrence C Paulson. 1994. *Isabelle: A generic theorem prover.* Springer.
[18] Kosta Serebryany. 2016. Continuous fuzzing with libfuzzer and addresssanitizer. In *2016 IEEE Cybersecurity Development (SecDev).* IEEE, 157–157.
[19] Julien Vanegue. 2022. Adversarial logic. In *International Static Analysis Symposium.* Springer, 422–448.
[20] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome logic: A unifying foundation for correctness and incorrectness reasoning. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 522–550.