

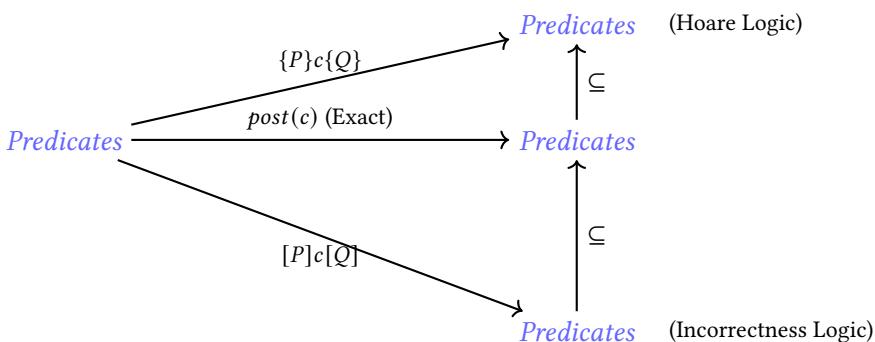
# 1 Skja: Adversarial Logic on Steroids

2 AKA SAI LALITH KUMAR, University of Colorado Boulder, USA

3 We introduce *Skja*, a new incorrectness logic [17] which extends on *Adversarial Logic* [21] with spi-calculus.  
4 Adversarial logic is very interesting in that it lets you apply under-approximation for exploitability analysis.  
5 The current core limitation of adversarial logic is that it only allows for parallel composition over multiple  
6 programs but it has problems with the model. With Skja, we bring the rich theory of spi-calculus to let the  
7 logic be composable for any infinite arbitrary set of programs which may be adversarial or not. This lets us  
8 encode and talk about attacks like Man-in-the-middle (MiTM) attacks and DDoS (Denial of Service) attacks.  
9 For this end, we first prove that parallel composition is a monoid over the adversarial logic grammar. We  
10 change the denotational semantics and program rules to support n-ary parallel composition while also proving  
11 that the changes do not effect the soundness. We also show a reasoning of Skja on Needham-Schroeder to  
12 highlight with usability in the MiTM attacks. We also give some future directions in this space.  
13

## 14 1 INTRODUCTION

15 When one reasons about programmes, we have access to a rich variety of logics, and one of  
16 them would be incorrectness logic [17] which is equivalent to [9]. The logic targets under-  
17 approximating the post of a condition  $c$ ; this is symmetrical to the work done by Hoare[12]  
18 which over-approximates. The way we can understand this through diagrammatic representation  
19 is by visualising the state space as a hierarchy of set inclusions, taken from O'Hearn's presentation.  
20 As shown in Figure 1, the central arrow represents the *strongest postcondition* ( $\text{post}(c)$ ), which  
21 describes exactly the set of states reachable by the program.  
22



35 Fig. 1. The hierarchy of program logic. Hoare logic (top) over-approximates the reachable states, while  
36 Incorrectness logic (bottom) under-approximates them.  
37

38 In this hierarchy, the top layer corresponds to standard Hoare Logic. It provides an *over-*  
39 *approximation*, meaning the postcondition  $Q$  must contain all reachable states ( $\text{post}(c) \subseteq Q$ ).  
40 This is useful for proving safety (i.e., "nothing bad happens").  
41

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

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:

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

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.

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

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  $Q \subseteq \text{post}(c)$ . Application of under-approximation to security reasoning is a very novel yet fundamental contribution [21]. 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 [20]. 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.

While Incorrectness Logic has shown great promise for sequential programs (finding memory bugs, null pointer dereferences), applying it to distributed security protocols presents a unique challenge[? ]. Protocols are not merely sequential programs; they are concurrent, message-passing systems operating in the presence of an active attacker (the Dolev-Yao model). Existing attempts to apply IL to security, such as Adversarial Logic [16], restrict the world to a binary interaction: "The Program" vs. "The Adversary." Skja removes this restriction. By marrying the under-approximate reasoning of IL with the Spi-calculus, a process calculus designed specifically for cryptographic protocols. We provide a logic that can reason about complex, multi-party interactions (Alice, Bob, Server, Attacker) without abstracting away the crucial details of channel mobility and distributed state. 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) [21] 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 and Privacy.** Standard AL models the world as a rigid couple  $(\sigma_p, \sigma_a)$ , representing a single global store for the program and a single store for the adversary. This forces a separation of concern that is unnatural for protocols. In a monolithic model, proving that Alice's private key remains secret requires complex side-conditions to ensure Bob (part of the "program") does not accidentally read it from the shared global heap. In reality, distributed systems share nothing but messages. Skja dissolves this monolith by mapping Process IDs to distinct local stores ( $\Pi(i)$ ). This allows us to verify not just that the *Attacker* does not know the secret, but that *Bob* does not know Alice's secret either, a requirement for privacy-preserving protocols that AL cannot easily express.

**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 [21], our work states what is new to the established semantics. We start by changing the state space. In AL, states were rigid couples  $(\sigma_p, \sigma_a)$ . To support  $n$ -ary composition, we map Process IDs to their local stores.

**DEFINITION 1 (GENERAL STATE SPACE).**  $\Sigma ::= [PIDs \rightarrow Variables \rightarrow Values]$

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  $\Delta$  as a mapping from PIDs to assertions.

**DEFINITION 2 (GENERALIZED TRIPLE).**  $[\Delta_{pre}]C[\Delta_{post}]$

For processes Alice ( $A$ ), Bob ( $B$ ), and Eve ( $E$ ),  $\Delta$  maps identities to assertions:  $\{A : P_A, B : P_B, E : P_{Eve}\}$ . The triple  $[\Delta]C[\Delta']$  is valid if, for all processes  $i \in \text{dom}(\Delta)$ , 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 [21].

148 DEFINITION 3 (SYNTAX).

149           Variables  $V ::= x \mid n \mid \alpha$

150           Expressions  $E ::= V \mid \text{rand}() \mid \text{enc}(E, K) \mid \langle E, E \rangle$

151            Assertions  $P, Q ::= E = E \mid \text{Knows}(E) \mid P \wedge Q \mid \exists x.P$

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

### 156 3.2 Semantics

157 DEFINITION 4 (SMALL-STEP SEMANTICS FOR COMPOSITION). Let  $C_1 \parallel \dots \parallel C_n$  be a composition of  
158  $n$  processes.

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

$$161 \quad \langle \Pi[i \mapsto \sigma_i], \Gamma \rangle \rightarrow \langle \Pi[i \mapsto \sigma'_i], \Gamma \rangle$$

- 163 • **Communication Step:** If process  $i$  writes to channel  $c$  and process  $j$  reads from channel  $c$ ,  
164 they synchronize:

$$165 \quad \langle \Pi, \Gamma \rangle \xrightarrow{i!c(v), j?c(v)} \langle \Pi', \Gamma' \rangle$$

167 where  $\Pi'$  updates the local variables of  $i$  and  $j$  appropriately.

- 168 • **Termination:**  $\text{skip} \parallel \dots \parallel \text{skip}$  is the terminal configuration.

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

170 HYPOTHESIS 1 (OWNERSHIP PRINCIPLE).

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

### 176 3.3 Inference Rules

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

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

$$182 \quad \frac{\text{N-PAR}}{\forall i \in \{1..n\}, \quad [\Delta_{\text{pre}}(i)]C_i[\Delta_{\text{post}}(i)]} \quad [\Delta_{\text{pre}}]C_1 \parallel \dots \parallel C_n[\Delta_{\text{post}}]$$

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

$$189 \quad \frac{\text{n-COM}}{[\Delta_{\text{pre}}(i)]\text{out}(c, v)[\Delta_{\text{post}}(i)] \quad [\Delta_{\text{pre}}(j)]\text{in}(c, x)[\Delta_{\text{post}}(j)] \quad \forall k \notin \{i, j\}, \Delta_{\text{pre}}(k) = \Delta_{\text{post}}(k)} \quad [\Delta_{\text{pre}}] (\dots \parallel \text{out}_i \parallel \dots \parallel \text{in}_j \parallel \dots) [\Delta_{\text{post}}]$$

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

$$195 \quad \exists v, (\Delta_{\text{pre}}(i) \implies x = v) \wedge (\Delta_{\text{post}}(j) \implies x = v)$$

### 197 3.4 The Dolev-Yao Adversary Model

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

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

$$\begin{array}{c} \frac{m \in \mathcal{K}}{\mathcal{K} \vdash m} \text{(Axiom)} \quad \frac{\mathcal{K} \vdash m_1 \quad \mathcal{K} \vdash m_2}{\mathcal{K} \vdash \langle m_1, m_2 \rangle} \text{(Pairing)} \quad \frac{\mathcal{K} \vdash \langle m_1, m_2 \rangle}{\mathcal{K} \vdash m_i} \text{(Proj)} \\ \\ \frac{\mathcal{K} \vdash m \quad \mathcal{K} \vdash k}{\mathcal{K} \vdash \{m\}_k} \text{(Encrypt)} \quad \frac{\mathcal{K} \vdash \{m\}_k \quad \mathcal{K} \vdash k^{-1}}{\mathcal{K} \vdash m} \text{(Decrypt)} \end{array}$$

211 In Skja, this deduction relation connects the operational semantics to the assertion logic. We  
 212 define the semantics of the assertion  $\text{Knows}(m)$  as follows:

$$213 \quad \sigma \models \text{Knows}(m) \iff \sigma(\text{knowledge}) \vdash m$$

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

## 222 4 PROOFS

### 223 4.1 Proof of Composition ( $\parallel$ ) forms a monoid

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

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

230 DEFINITION 5 (STRUCTURAL CONGRUENCE). *The parallel composition operator  $\parallel$  satisfies the*  
 231 *following laws:*

$$233 \quad P \parallel Q \equiv Q \parallel P \quad (\text{Commutativity}) \quad (1)$$

$$234 \quad (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \quad (\text{Associativity}) \quad (2)$$

$$235 \quad P \parallel \text{skip} \equiv P \quad (\text{Identity}) \quad (3)$$

237 THEOREM 1. *(Process,  $\parallel$ , skip) form a commutative monoid.*

238 Proof: A commutative monoid has three properties for it to satisfy. We show that ( $\parallel$ ) satisfies all  
 239 below:

- 241 • **Commutativity:** For some processes  $P$  and  $Q$ ,  $P \parallel Q \equiv Q \parallel P$ . Every derivation of a step  
 242 from  $P \parallel Q$  is obtained by applying one of the small-step rules. Swapping the syntactic  
 243 positions of  $P$  and  $Q$  gives a derivation of the same step from  $Q \parallel P$  (process IDs are intrinsic  
 244 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,  $(P \parallel \text{skip}) \equiv P$ . 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  $\{C_1, \dots, C_n\}$ , 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 [21] to our  $n$ -ary state space  $\Sigma$ .

LEMMA 2 (CHARACTERIZATION). *A triple  $[\Delta_{\text{pre}}]C[\Delta_{\text{post}}]$  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:*

$$\forall \Pi' \in \llbracket \Delta_{\text{post}} \rrbracket, \exists \Pi \in \llbracket \Delta_{\text{pre}} \rrbracket \text{ such that } \langle \Pi, \Gamma \rangle \xrightarrow{*} \langle \Pi', \Gamma' \rangle$$

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

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

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  $\forall i, [\Delta_{\text{pre}}(i)]C_i[\Delta_{\text{post}}(i)]$  hold, then  $[\Delta_{\text{pre}}]C_1 \parallel \dots \parallel C_n[\Delta_{\text{post}}]$  satisfies the Characterization Lemma.

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

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

$$\forall i, \exists \sigma_i \in \llbracket \Delta_{\text{pre}}(i) \rrbracket \text{ such that } \langle \sigma_i, \Gamma \rangle \xrightarrow{*} \langle \sigma'_i, \Gamma'_i \rangle$$

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

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

Because the domains of  $\sigma_i$  are disjoint (Hypothesis 1), the local transitions of each process  $C_i$  can be interleaved without interference. Therefore, the sequence of independent steps transforms  $\Pi$  into  $\Pi'$ . Since  $\Pi \in \llbracket \Delta_{\text{pre}} \rrbracket$  and  $\Pi' \in \llbracket \Delta_{\text{post}} \rrbracket$ , the characterization holds.  $\square$

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  $\Pi' \in \llbracket \Delta_{\text{post}} \rrbracket$ . We focus on the receiver's state  $\sigma'_j$  (the sender's side follows similarly).

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

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

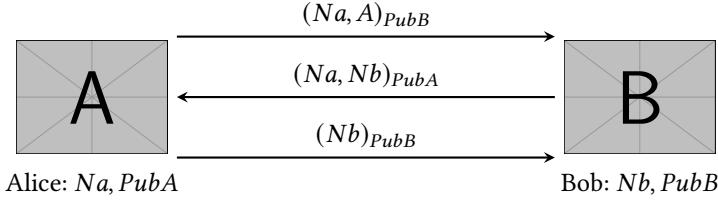


Fig. 2. Needham-Schroeder

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  $\sigma_j$  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.  $\square$

## 5 REASONING WITH SJKA: NEEDHAM-SCHROEDER

### 5.1 Needham-Schroeder

Needham-Schroeder(NS)[16], is an authentication protocol. The way it runs is as follows:

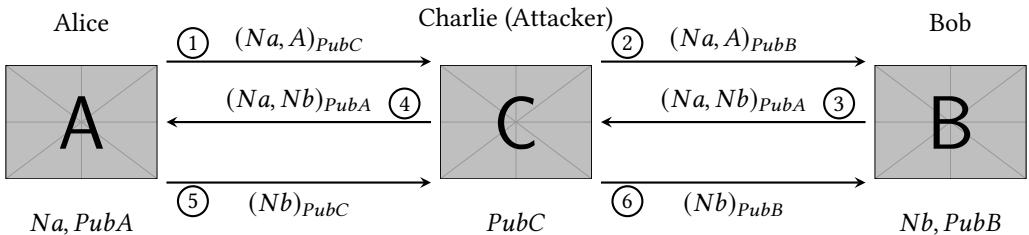


Fig. 3. Lowe's man-in-the-middle attack, where Charlie impersonates Alice.

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

344 for Bob it's broken. This can be observed at the end of the protocol run where Bob thinks he is  
 345 talking to Alice, but it's actually Charlie who now has Bob's and Alice's nonce, so Charlie can act  
 346 as Alice and keep the conversation going. This is a failure of authentication.

## 347 348 5.2 Formalizing the Attack in Skja

349 We demonstrate the validity of Lowe's attack by tracing the evolution of the generalized assertion  
 350 map  $\Delta$  through the application of the **n-Com** rule.

351 Let the initial state be  $\Delta_0$ . The participants have the following initial knowledge:

$$352 \quad \Delta_0 = \{A : \text{Knows}(Na, PrivA), B : \text{Knows}(Nb, PrivB), C : \text{Knows}(PrivC)\}$$

### 353 354 Phase 1: Initiation and Impersonation (Steps 1 & 2)

355 Alice intends to talk to Charlie. She executes  $\text{out}(c, \{Na, A\}_{PubC})$ . Charlie reads from  $c$ . Applying  
 356 **n-Com**:

$$357 \quad [\Delta_0(A)]\text{out} \cdots \parallel \text{in}[\Delta_1(C)]$$

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

$$360 \quad \Delta_1(C) \implies \text{Knows}(Na)$$

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

$$363 \quad \Delta_2(B) \implies \text{Received}(\{Na, A\}_{PubB})$$

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

### 365 366 Phase 2: The Oracle Step (Steps 3 & 4)

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

$$368 \quad \Delta_3(C) \implies \text{Knows}(\{Na, Nb\}_{PubA})$$

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

$$373 \quad \Delta_4(A) \implies \text{Received}(\{Na, Nb\}_{PubA})$$

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

### 376 Phase 3: The Leak (Step 5)

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

$$380 \quad [\Delta_4(A)]\text{out}(c, \{Nb\}_{PubC}) \parallel \text{in}(c, x)[\Delta_5(C)]$$

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

$$382 \quad \Delta_5(C) \implies \text{Knows}(Nb)$$

384 The adversary has successfully extracted the secret nonce  $Nb$ .

### 385 Phase 4: Completion (Step 6)

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

### 388 Conclusion

389 The final state  $\Delta_{final}$  contains a contradiction to the authentication specification:

$$390 \quad (1) \Delta_{final}(B) \implies \text{Partner} = Alice$$

$$391 \quad (2) \Delta_{final}(C) \implies \text{Knows}(Na, Nb)$$

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

## 396 397 6 RELATED WORK

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

401 **Program Logics.** The foundation of our work rests on Incorrectness Logic [17], which in-  
 402 verted the reasoning of standard Hoare Logic [11, 12]. While standard logic proves the absence of  
 403 bugs, incorrectness logic proves their presence. This is particularly relevant for security, where  
 404 demonstrating an exploit path is often more valuable than a theoretical safety guarantee.

405 **Process Calculi and Security.** The use of process calculi [15] to model security protocols was  
 406 pioneered by Abadi and Gordon with the spi-calculus. This introduced cryptographic primitives  
 407 directly into the operational semantics of the  $\pi$ -calculus. While automated tools like ProVerif [4],  
 408 WALDO [13] and Tamarin[14] have successfully applied these concepts using model checking, Skja  
 409 aims to bring this expressivity into a deductive logic framework.

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

415 As noted by O’Hearn [17] and Vanegue [21], incorrectness logic can be viewed as an abstract  
 416 interpretation where the domain  $\mathcal{D}$  represents subsets of reachable states. For Skja, our domain  
 417 is enriched with the Dolev-Yao adversary model. We can define our abstract domain  $\mathcal{D}^\#$  as the  
 418 product of the program state lattice and the adversary knowledge lattice:

$$419 \quad 420 \quad \mathcal{D}^\# = \mathcal{P}(\Sigma_{prog}) \times \mathcal{K}_{adv}$$

421 where  $\mathcal{K}_{adv}$  represents the set of derivable messages. The abstraction function  $\alpha$  maps concrete  
 422 execution traces to these logical assertions. Our fixpoint computation, therefore, does not find the  
 423 "smallest set containing all behaviors", but rather attempts to find a "non-empty set of confirmed  
 424 behaviors". This aligns Skja with *under-approximate abstract interpretation*. There is also rather  
 425 interesting working showing the connection between abstract interpretation and program logics  
 426 which state the connection between them [7].

427 **Adversarial Logic.** The most direct predecessor to Skja is Adversarial Logic [21]. While ground-  
 428 breaking in applying under-approximation to security, it is limited by its "rigid couple" state space  
 429 (Program vs. Adversary). Our work generalizes this to an  $n$ -ary composition, effectively unblocking  
 430 the modeling of multi-party protocols and complex topologies that were previously awkward to  
 431 represent in the original Adversarial Logic framework.

## 432 433 7 CONCLUSION AND FUTURE WORK

434 In this paper, we adapted under-approximation logic for security and boosted its versatility by  
 435 integrating techniques from the spi-calculus. This in effect turned the adversarial logic from a rigid  
 436 monolith to a dynamic system. We demonstrated that this extension preserves the soundness of the  
 437 underlying logic through formal proofs. Finally, we applied Skja to the Needham-Schroeder protocol,  
 438 formally proving the validity of the classic Man-in-the-Middle vulnerability. The expressibility of  
 439 Skja shines when it comes to multi-parties, especially in the context of IOT and distributed systems.  
 440 One strong fit we can see is to use Skja to reason about Denial of Service Attacks (DDoS). The

442 spi-calculi based !P operator would be the right fit. An interesting line of future work is to turn  
 443 Skja into an algorithm which not only verifies but synthesizes the attack traces. This could be  
 444 achieved through a bounded model checking based search algorithms. Both under-approximation  
 445 and over-approximation could be combined like in [22] to get benefit of both worlds. We believe  
 446 this to be an interesting research direction. One could also Certify incorrectness logic and Skja  
 447 with a dependent type theory based theorem prover like Lean [8], ROCQ [5] and Isabelle[18]. The  
 448 Dolev-Yao attacker formalism could follow the work [19] which follows Isabelle to construct a  
 449 induction based protocol. Finally, as reviewed in related work, application of abstract interpretation  
 450 to underapproximation is exciting direction.

## 451 452 ACKNOWLEDGMENTS

453 I thank my advisor, Dr. Gowtham Kaki. I acknowledge Dr. Bor-Yuh Evan Chang and Kirby Linvill  
 454 for teaching me all I know about PL. Thanks to Dakota Brayan for feedback, and Dr. Fabio Somenzi  
 455 for teaching me about logic.  
 456

## 457 REFERENCES

- 459 [1] Nahid A Ali. 2017. A survey of verification tools based on hoare logic. *International Journal of Software Engineering & Applications* 8 (2017), 87–100.
- 460 [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.
- 461 [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.
- 462 [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.
- 463 [5] Adam Chlipala. 2013. *Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant*. MIT Press.
- 464 [6] Patrick Cousot. 1996. Abstract interpretation. *ACM Computing Surveys (CSUR)* 28, 2 (1996), 324–328.
- 465 [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.
- 466 [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.
- 467 [9] Edsko De Vries and Vasileios Koutavas. 2011. Reverse hoare logic. In *International Conference on Software Engineering and Formal Methods*. Springer, 155–171.
- 468 [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.
- 469 [11] Robert W Floyd. 1993. Assigning meanings to programs. In *Program Verification: Fundamental Issues in Computer Science*. Springer, 65–81.
- 470 [12] Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580.
- 471 [13] 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.
- 472 [14] 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.
- 473 [15] Robin Milner. 1980. *A calculus of communicating systems*. Springer.
- 474 [16] Roger M Needham and Michael D Schroeder. 1978. Using encryption for authentication in large networks of computers. *Commun. ACM* 21, 12 (1978), 993–999.
- 475 [17] Peter W O’Hearn. 2019. Incorrectness logic. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–32.
- 476 [18] Lawrence C Paulson. 1994. *Isabelle: A generic theorem prover*. Springer.
- 477 [19] Lawrence C Paulson. 1997. Proving properties of security protocols by induction. In *Proceedings 10th Computer Security Foundations Workshop*. IEEE, 70–83.
- 478 [20] Kosta Serebryany. 2016. Continuous fuzzing with libfuzzer and addresssanitizer. In *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 157–157.
- 479 [21] Julien Vanegue. 2022. Adversarial logic. In *International Static Analysis Symposium*. Springer, 422–448.

- 491 [22] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome logic: A unifying foundation for correctness  
492 and incorrectness reasoning. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 522–550.
- 493
- 494
- 495
- 496
- 497
- 498
- 499
- 500
- 501
- 502
- 503
- 504
- 505
- 506
- 507
- 508
- 509
- 510
- 511
- 512
- 513
- 514
- 515
- 516
- 517
- 518
- 519
- 520
- 521
- 522
- 523
- 524
- 525
- 526
- 527
- 528
- 529
- 530
- 531
- 532
- 533
- 534
- 535
- 536
- 537
- 538
- 539