

Bindings for Security Protocol Message Composition

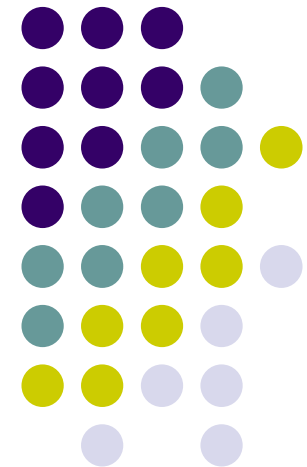
Genge Bela¹, Haller Piroska²



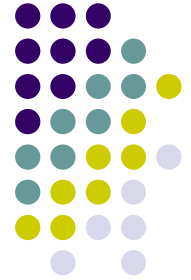
¹ Junior lecturer at the “Petru Maior” University of Târgu Mures and
a PhD student at the Technical University of Cluj-Napoca

² Lecturer at the “Petru Maior” University of Târgu Mures, PhD

Contact: {¹bgenge, ²phaller}@upm.ro



Summary



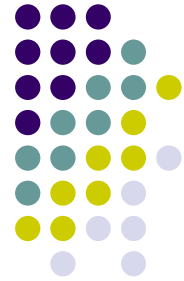
- Introduction to security protocols and composition
- Requirements for message composition
- Existent protocol specifications
- The strand space model
- Extending the strand space model with *bindings*
- Composition
- Creating a new protocol from two existing security protocols
- Conclusion and future work



Introduction

- Security protocols?
 - Communication protocols + cryptography
 - Example message specification: $\{A, B, Na, K\}_{Kab}$
- Composition?
 - Combining two or more protocols
 - Useful in design process
 - Composition types: *sequential* and *parallel*
- Sequential composition?
 - Protocols are run sequentially
 - Verifying the independence of the involved protocols
- Parallel composition?
 - Unifying protocol roles
 - Message composition additional requirement: determining optimal message structure

Compositionality requirements for *parallel* composition



- Protocol roles are merged together
- Goal:
 - Create homogenous protocols with multiple security properties
 - Homogenous: initial protocol messages can not be clearly distinguished
- Requirements:
 - Security properties for each protocol must be maintained:
 - Maintain message pre-conditions: message sequence
 - Maintain cryptographic context for each component: context can not be changed to be weaker, but only stronger
 - Verifying if the resulting messages respect the “strong independence” property
- Specification requirements:
 - Explicit message component-link specification
 - Explicit message sequence specification

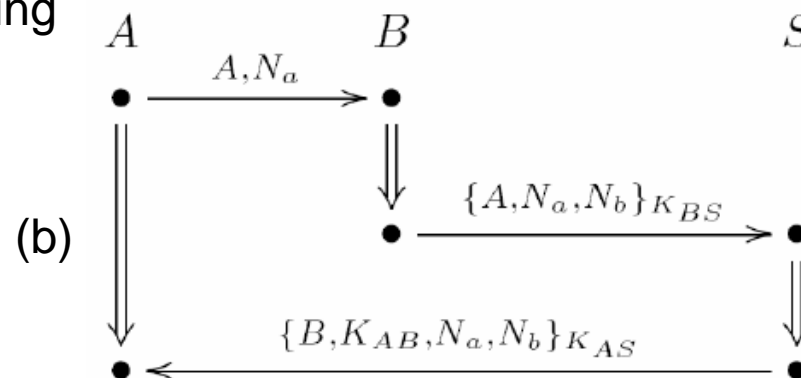
Existent specification models



- Existing message specification models are not satisfying because:
 - They are simplistic
 - Message specification does not contain sufficient information for formal reasoning
- Informal specification (a):
 - Inexistent formal reasoning language
- Strand space model (b):
 - Provides a formal reasoning language
 - Informal message specification
- Operational semantics (c):
 - Message components are made explicit
 - Message-term links are not made explicit

(a)

$A \rightarrow S: A, \{T_a, B, K_{ab}\}K_{as}$
 $S \rightarrow B: \{T_s, A, K_{ab}\}K_{bs}$
 $B \rightarrow A: \{N_b\}K_{ab}$
 $A \rightarrow B: \{N_{b+1}\}K_{ab}$



(c)

$$\begin{aligned}
 ns(i) = & (\{i, r, ni, sk(i), pk(i), pk(r)\}, \\
 & send_1(i, r, \{i, ni\}_{pk(r)}) \cdot \\
 & read_2(r, i, \{ni, V\}_{pk(i)}) \cdot \\
 & send_3(i, r, \{V\}_{pk(r)}))
 \end{aligned}$$

Why extending the strand space model?



- Strand:
 - A sequence of send and receive events
 - Model protocol participants
 - Model intruder capabilities: Concat, Split, Replay, Encrypt, Decrypt
- Strand spaces:
 - A collection of strands
 - Model security protocols
- Why extending the strand space model?
 - Highly general specification
 - Allows the modeling of internal mechanisms: encryption, decryption, key generation
 - Flexibility



Specializing basic sets

- Explicit modeling of function names:

| | | |
|----------------|-----|---------------|
| $FuncName ::=$ | sk | (secret key) |
| | pk | (public key) |
| | pvk | (private key) |
| | h | (hash) |

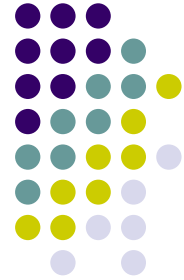
- Explicit term construction:

$$\mathcal{T} ::= . \mid R \mid N \mid K \mid (\mathcal{T}, \mathcal{T})$$

$$\mid \{\mathcal{T}\}_{FuncName(\mathcal{T})}$$

- The resulting strand model: $\langle \pm t_1, \pm t_2, \dots, \pm t_n \rangle \in (\pm \mathcal{T})^*$

$$\left. \begin{array}{l} n_1 = \langle s, i \rangle \\ n_2 = \langle s, i+1 \rangle \\ strand(n_1) = strand(n_2) \end{array} \right\} n_1 \Rightarrow n_2 \quad \left. \begin{array}{l} n_1, n_2 \in \mathcal{N} \\ strand(n_1) \neq strand(n_2) \end{array} \right\} n_1 \rightarrow n_2$$



Adding binding terms

- Message types (i.e. patterns):

$$\begin{array}{l}
 \textit{BasicTT} ::= r \quad (\textit{role type}) \\
 \quad | n \quad (\textit{nonce type}) \\
 \quad | k \quad (\textit{key type}) \\
 \quad | b \quad (\textit{binding type}) \\
 \quad | m \quad (\textit{user-defined message})
 \end{array}$$

- Basic term: $\langle \rho, \nu, \kappa, \beta, f, k, \theta \rangle$

$$\rho \in R^* \quad \nu \in N^* \quad \kappa \in K^* \quad \beta \in B^* \quad f \in \textit{FuncName} \quad k \in K \quad \theta \in \textit{BasicTT}^*$$

- Function mappings and classifiers:

$$\textit{Roles}(b) = \rho, \textit{Nonces}(b) = \nu, \textit{Keys}(b) = \kappa, \quad C ::= C_{\mathcal{R}} \quad (\textit{Role classifier})$$

$$\textit{Bindings}(b) = \beta, \textit{Func}(b) = f, \textit{BindingKey}(b) = k, \quad | C_{\mathcal{K}} \quad (\textit{Knowledge classifier})$$

$$\textit{TypeSet}(b) = \theta$$

- Introducing b-strands: $s : (\pm B)^* \times C$

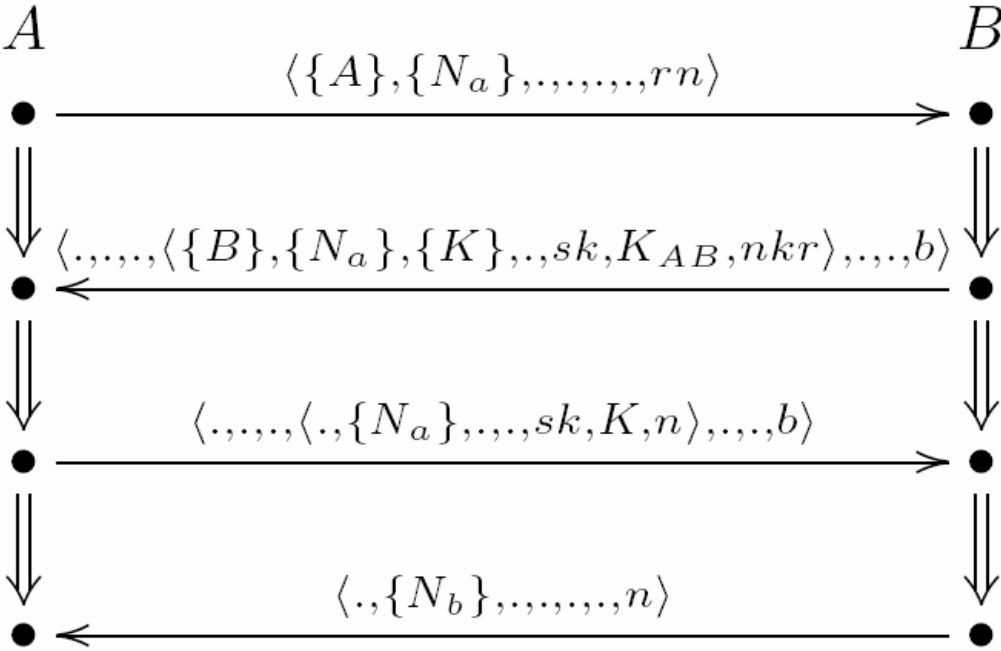
B-strand specification example



- Lowe’s modified “BAN concrete Andrew Secure RPC”

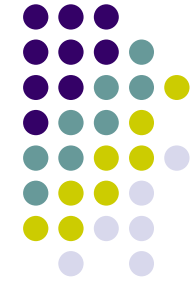
$A \rightarrow B: A, N_a$
 $B \rightarrow A: \{N_a, K, B\}_{K_{AB}}$
 $A \rightarrow B: \{N_a\}_K$
 $B \rightarrow A: N_b$

Regular specification



B-strand-based specification

Composition algorithm



- Construct all possible message combinations
- At each step:
 - Message pre-conditions are verified
 - The performance of the resulting protocol is evaluated
- Finally:
 - Verify message component “strong independence (SI)” property:
 - SI: structural knowledge-based similarity analysis (proposed in a previous work)

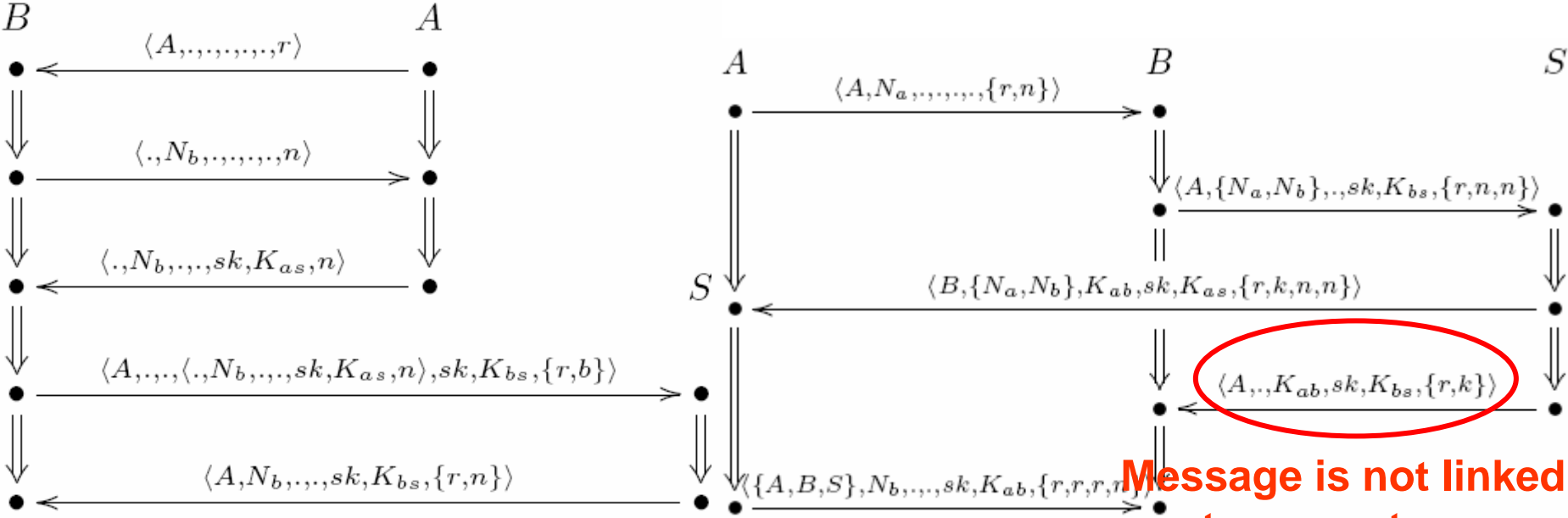
```

FOR EACH  $n_2 \in (getBStrand(r_{s2}, C_{\mathcal{R}}))_1$ ,
     $n_1 \in (getBStrand(r_{s1}, C_{\mathcal{R}}))_1$ ,
         $sign(n_2) = +, sign(n_1) = +$ 
         $b_1 = binding(n_1)$ 
         $b_2 = binding(n_2)$ 
        IF  $n_2 \notin ProcNodes$  AND
             $Func(b_2) \neq 'kenc'$  AND
             $Func(binding(n_1)) \neq 'kenc'$  AND
             $destRole(n_2) = destRole(n_1)$  AND
             $accumKnow(n_2) \sqsubseteq accumKnow(n_1)$  AND
             $predNodeSeq(n_2) \subseteq predNodeSeq(n_1)$ 
             $b_1 = b_1 + b_2$ 
             $accumKnow(destNode(n_1)) =$ 
                 $accumKnow(destNode(n_1)) +$ 
                 $accumKnow(n_2)$ 
             $ProcNodes = ProcNodes \cup \{n_2\}$ 
        ENDIF
    ENDFOR
    
```

Parallel composition example



- Woo and Lam Pi 3
 - One-way authentication
- Lowe's modified Yahalom
 - Two way authentication
 - Key exchange
 - One of the authentication messages is flawed

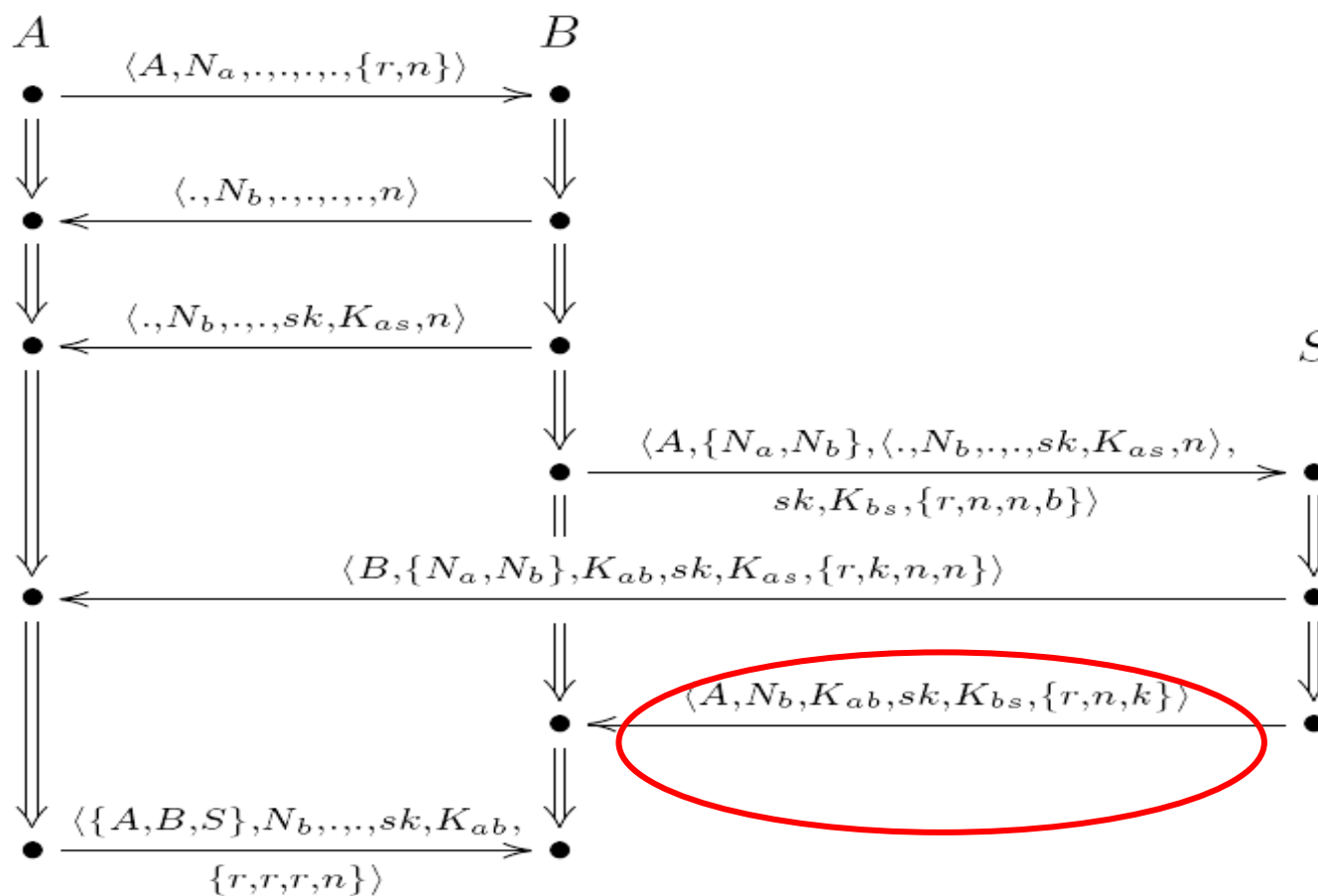


Message is not linked to current run (does not contain Nb)



Resulting protocol

- Solves the authentication problem





Future work

- Add performance-related information to optimize the performance of the created protocols
- Implement a tool for the automated composition of security protocols

Thanks for your attention!



- Questions?