Security Protocols as Choreographies

Bruni, Alessandro; Carbone, Marco; Giustolisi, Rosario; Mödersheim, Sebastian Alexander; Schürmann, Carsten

Published in:
Protocols, Strands, and Logic

Link to article, DOI:
10.1007/978-3-030-91631-2_5

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Security Protocols as Choreographies

Alessandro Bruni[1], Marco Carbone[1(✉)], Rosario Giustolisi[1], Sebastian Mödersheim[2], and Carsten Schürmann[1]

[1] IT University of Copenhagen, 2300 Copenhagen S, Denmark
{brun,carbonem,rosg,carsten}@itu.dk
[2] DTU Compute, 2800 Lyngby, Denmark
samo@dtu.dk

Abstract. A choreography gives a description of how endpoints in a concurrent systems should exchange messages during its execution. In this paper, we informally introduce a choreographic language for describing security protocols and a property language for expressing non-trivial security properties of such protocols. We motivate this work using the envelope protocol [2] as an example, which ensures auditable transfers by means of a TPM, that guarantees that the issuer of a message always learns whether such message has been opened or not. We then take an implementation of the TPM formulated as an API and discuss how such implementation and the usage of the TPM in the protocol can be related. Finally, we illustrate how the protocol and property descriptions can be translated into multiset rewrite rules and metric first order logic respectively, in order to check if auditable transfer holds.

Keywords: Security protocols · Choreography · Verification

1 Introduction

Choreographic programming [13,14,34] is a programming paradigm for concurrent systems that focuses on the global flow of interactions that communicating peers are supposed to follow during execution rather than their local sequence of send and receive operations. Choreographies have been studied extensively in the context of concurrency theory and programming languages, but they have only been sporadically considered for modeling security protocols [6,11,12]. This is quite surprising, because Alice-Bob notations, which are prevalently used in security theory [3,10,27,30], are closely related to choreographies that describe the communication structure of entire systems. In fact, extensions of the Alice-Bob notation with more features such as long-term state or subprotocols [7] can be found in the literature.

In this paper, we celebrate Joshua Guttmann by proposing a choreography language extended with term algebras and equational theories for modelling various cryptographic primitives used in security protocols, for example, encryption and decryption, signatures and verifications, etc. The main idea is that security

D. Dougherty et al. (Eds.): Guttman Festschrift, LNCS 13066, pp. 98–111, 2021.
https://doi.org/10.1007/978-3-030-91631-2_5

protocols can be written in such language and then the minimum local behavior of each participant can be automatically generated from the choreography through an operation called *endpoint projection*. Note that a choreography *only* provides the local behavior of *honest* participants (while the intruder may not stick to the protocol and is rather defined, for instance, by a Dolev-Yao-style model). Generating the endpoint projection requires an analysis of the honest agents knowledge, and how they can compose and decompose the messages that they send and receive [3,30]. Since this problem is well-understood, here we focus on those aspects of choreographic languages that distinguish them from traditional Alice-and-Bob narrations. The (honest) local behaviour generated from a choreography specifying a protocol can then be used for two purposes. First, it can be used as a local specification for verifying that a given implementation is compliant with a given API (similarly to what is done for multiparty session types [23]), e.g., in the TPM envelope example reported in this article, where we check that the usage of the TPM in the protocol description is compatible with the TPM API. Second, it allows us to replace the local projection of API-like participants with their API implementations, and use them along with the local projections of the remaining honest agents as the input for a protocol analyser, where we can verify some security properties. A key feature of our approach is that such properties can be specified at the choreographic level and then automatically translated, in a semantic preserving way, into the language of the protocol analyser automatically, via the endpoint projection.

This paper is not a theory paper in that it does not provide a formal development of the translation of choreographies. It should be read as a position paper that elaborates the idea of choreographies applied to security protocols and their endpoint projections by the means of an example, namely the *envelope protocol*, first proposed by Ables and Ryan [2] and analyzed by Delaune et al. [17]. Joshua Guttman and colleagues [22] proposed an extension to the CPSA tool [18] to protocols with state, contributing to the first verification of the envelope protocol with unbounded reboots, while also introducing a modular approach that faithfully represents the interface offered by the trusted third party used in the protocol.

The envelope protocol uses the Trusted Platform Module (TPM) as a trusted party to guarantee a security property that we have dubbed *auditable transfer*: Alice wants to share a message with Bob such that Alice will be able to verify if Bob has opened the message or not. The TPM is used to securely store the state of this transfer.

In this paper, we give a formulation of the auditable transfer protocol as a choreography in Sect. 2, describe the result of the endpoint projection including the property in Sect. 3, and feed the result into Tarmarin to verify the auditable transfer property in Sect. 4. Our main goal with this notation is to allow for a simple, clear and yet very expressive specification language. We revisit related work in Sect. 5 and assess results and outline future work in Sect. 6.

2 The Envelope Protocol and Its Choreographic Description

The envelope protocol aims at being the digital version of a sealed physical envelope. A sealed envelope allows to achieve auditable transfer: the recipient can either obtain the content inside the envelope or prove that they have not broken the seal and thus obtained the content, with no intervention required from the sender. In this paper, we use the classic example of the envelope protocol: Alice would like her parents to know where she is going out for the night only in case of necessity, so she writes this information in a letter and puts it into a sealed envelope. Alice does not necessarily trust her parents, who may behave adversarially and hence may like to know where Alice is going out without her noticing that they learned this information. The sealed envelope protects Alice in this case, as learning the information requires breaking the seal, which Alice would notice upon her comeback. Its digital counterpart cannot be implemented with cryptography alone, as revealing the content of a message usually requires obtaining the key or solving an interactive challenge.

The envelope protocol relies on a trusted third party, the TPM, which provides an interface to create a key that is concealed inside the trusted computing module, and offers functions for encryption and decryption which are bound to the internal state of the TPM. The internal state of a TPM is made of 24 Platform Configuration Registers (PCRs), which essentially implement a hash chain: the TPM allows to reset one of these registers to an initial known value (with the `Boot` command) and to extend the chain with a new value (with the `Extend, n` command). For simplicity we assume only one register, which is also what the envelope protocol requires, hence all the commands omit the first parameter. The three other commands that the TPM implements that are used by the envelope protocol are the following: the `Create, s` command that creates a new public-private key bundle and releases the public key (the corresponding private key is retained by the TPM and can only be used through the TPM interface when the `pcr` is in state `s`); the `Quote, x` command that binds the message `x` to the current value of the `pcr`, and the `Decrypt` command that takes a ciphertext and a key bundle and returns the decryption only if the state of the key bundle matches the current value of the `pcr`. In Sect. 3.2, we formalise the TPM interface and show that the envelope protocol respects this interface.

The envelope protocol uses the TPM as follows: Alice first resets the TPM to its initial `pcr` state `1` and then issues the `Extend` command with a fresh secret nonce `n`, so the TPM is now in state `hash(n,1)`. Now Alice asks the TPM to create a public key `k` for the state `hash(obtain, hash(n, 1))`, i.e., the TPM will only decrypt messages using `inv(k)`, when the TPM is in that state. The TPM can be brought into that state by performing the `Extend` command with value `obtain`, but afterwards it is impossible to bring it back to state `hash(n,1)` unless one knows the nonce `n`. Alice now encrypts her message `v` with `k` and sends it to her parent. The parent now has two options: they can either extend the PCR to the state `hash(refuse, hash(n, 1))` and obtain a proof that they refused to open

```
Roles: Alice, TPM[Honest], Parent
Knowledge: tpmk

Protocol
  Alice → TPM: Boot
  TPM: pcr := '1'
  TPM → Alice: Booted
  Alice: new n, new esk
  Alice → TPM: Session, tpmk, aenc(esk, tpmk)
  TPM: new sid
  TPM → Alice: sid
  Alice → TPM: senc((Extend, n, sid), esk)
  TPM: pcr := hash(n, pcr)
  TPM → Alice: Extended
  Alice → TPM: Create, hash('obt', pcr)
  TPM: new k
  TPM → Alice: sign((Created, k, hash('obt', pcr)), inv(tpmk)
    )
  Alice: new v
  Alice → Parent: Envelope, enc(v, k)
  Parent: new esk
  Parent → TPM: Session, aenc(esk, tpmk)
  TPM: new sid
  TPM → Parent: sid
  Parent → TPM: {
    senc((Extend, 'ref', sid), esk):
    TPM: pcr := hash('ref', pcr)
    TPM → Parent: Extended
    Parent → TPM: Quote, enc(v, k)
    TPM → Parent: sign((Quoted, pcr, enc(v, k)), inv(tpmk))
    event secret(v)
  +
    senc((Extend, 'obt', sid), esk):
    TPM: pcr := hash('obt', pcr)
    TPM → Parent: Extended
    Parent → TPM: Decrypt, enc(v, k), sign((Created, k, pcr),
     inv(tpmk))
    TPM → Parent: v
  }

Objectives
  Intruder learns v implies not secret(v)
```

Fig. 1. The envelope protocol as a chreography

the letter, or they can extend the PCR to the state `hash(obtain, hash(n, 1))` and use the TPM to decrypt Alice's message.

We now give a precise specification of the envelope protocol as a choreography, which is depicted in Fig. 1. A choreography consist of four sections: `Roles`, `Knowledge`, `Protocol`, and `Objectives`. The roles and initial knowledge declared

are standard. More interesting is the specification of the protocol. The TPM offers different services, such as `Boot`, `Extend`, `Create`, `Quote`, `Decrypt`, or `Envelope`, which are followed by their respective parameters. Upon completion, the TMP signals the caller that a particular service has terminated, again using messages, such as `Booted`, `Extended`, `Created`, `Quoted`, or `Decrypted`. We use M to denote messages.

Choreographies also support state. In our example the state is denoted by `pcr`, the internal state of the TPM. The expression algebra that we use, includes operations such as bit string concatenation, denoted by a comma, hashing, denoted by `hash`, and encryption, denoted by `enc`. Dereferencing `pcr` is a silient operation. Expressions are denoted by E.

The language supports two forms of command, generically denoted by C: a command to create fresh nonces written as `new`, and another command for assignment, denoted by `:=`. The scope of nonces extends to the end of the protocol specification, but a priori, only the principal who creates the nonce knows it. Let A and B be two different roles. Protocols P, Q are defined by a sequence of operations, in Alice Bob notation, message transfer of message M as $A \to B : M$, internal execution of command C as $A : C$, and choice $A \to B : \{P + Q\}$.

As part of a formal semantics in the style of [3,11,30], we rule out as *not executable* (or *not well-formed*) those specifications that require participants to produce messages that they actually cannot produce (without breaking cryptography). This, however, requires a considerable amount of formal machinery that we do not want to introduce in this more conceptual paper.

3 Projection and Refinement

In the previous section, we have shown how to use a choreographic language for specifying the envelope protocol. Besides proving the correctness of such a protocol (which we will do in the next section), we show how we can use a type-like approach for checking that an implementation of (some of) the participants is compliant with the behavioural specification given by the protocol. In order to do so, we proceed by two steps. First, we define the notion of *projection*, a well-studied concept in the theory of choreographies.

3.1 Projection

The projection of a choreography with respect to a particular endpoint is a specification of how such endpoint has to behave in the protocol. In a nutshell, given the choreography $A \to B : M_1; B \to A : M_2$ for example, the projection with respect to A is `send` M_1 `;` `receive` M_2, while the projection with respect to B is `receive` M_1`;` `send` M_2, were `send` and `receive` are standard endpoint operations, i.e. commands. However, in general just literally taking the messages from the choreography for the endpoint actions will not be correct, e.g., if M_1 is an encrypted message that B cannot decrypt, then it must be replaced by a variable as first observed by Lowe [27]. This question is in general also related

to the algebraic properties of cryptographic operators that we consider, e.g., the properties of exponentiation in Diffie-Hellman. In general such a formal semantics can be given in the style of [3]. The endpoint specification is useful, because we can use it for checking that, e.g., a given implementation of A follows the specification given by the original choreography.

In the envelope protocol, the behaviour of `Alice` according to the choreographic specification of the envelope protocol consists of:

```
Role Alice
 send Boot
 new n
 send Extend(n)
 receive Extended
 send Create(hash('obt', pcr))
 receive Created(k, hash('obt', pcr))
 new v
 send Envelope(enc(v, k))
```

The projection of `Alice` corresponds to her behaviour in the choreography. Above, `new n` creates a fresh nonce `n` and works as a binder in the subsequent code. On the other hand, `send Extend(n)` sends a message which selects option `Extend` and also communicates the value `n`, which in this case is bound by the `new n` in the second line. When receiving a message, the language uses structured terms with constants and variables, implying standard pattern matching. Similarly, we can project the behaviour of the TPM:

```
Role TPM
 receive Boot
 pcr := '1'
 receive Extend(n)
 send Extended
 receive Create(hash('obt', pcr))
 new k
 send Created(k, hash('obt', pcr))
 receive {
    Extend('ref'):
      pcr := hash('ref', pcr)
      send Extended
      receive Quote(enc(v, k))
      send Quoted(pcr, enc(v, k))
    +
    Extend('obt'):
      pcr := hash('obt', pcr)
      send Extended
      receive Decrypt(enc(v, k)), Created(k, pcr)
      send Decrypted(v)
}
```

In the case of TPM, we note that the receive operation can also handle choice. Options in the choice are separated by the keyword + as in standard choreographies which corresponds to the standard external choice from the pi-calculus [29]. As mentioned in the previous sections, in order to ensure consistency of these specifications, each branch must have a unique label. In order to achieve this, while retain flexibility, we use pattern matching to distinguish branches with the same label. E.g., above, although the label `Extend` is identical in both branches, each branch can be identified by the constant that the branch is expecting to receive. Finally, this is the projection of the parent's expected behaviour:

```
Role Parent
 receive Envelope(enc(v, k))
 send {
     Extend('ref'):
       receive Extended
       send Quote(enc(v, k))
       receive Quoted(pcr, enc(v, k)) (1)
 +
     Extend('obt'):
       receive Extended
       send Decrypt(enc(v, k)), Created(k, pcr)
       receive Decrypted(v)
 }
```

Dually to the external choice provided by the TPM, the parent's projection is making an internal choice: it either sends the `ref` value or the `obt` value.

3.2 Refinement

In the theory of choreographies, choreographic specifications are projected into endpoints behaviour. Such cut of the global behaviour can often be used by a type system to do a local type checking of code. In here, our choreographies are richer in the sense that contain information about values that the protocol being described should handle. Hence, the specification and a possible implementation are very close. In this subsection, we illustrate how a notion of refinement could be used for verifying that an implementation is compliant to the protocol specification. In order to do so, we focus on the TPM behaviour. Obviously, the projection from the choreography given above does not have to be the exact way the TPM should be implemented. In general, a TPM is a piece of hardware that can provide the TPM service to system components. Therefore, it is usually implemented as a simple API. The one below is a possible API implementation:

```
TPM(pcr) = {
    receive Boot: {
        send Booted;
        TPM(-1)
    } +
```

```
    receive Create, s: {
        new k
        send sign((Created, k, s), inv(tpmk))
        TPM(pcr)
    } +
    receive Quote, x: {
        send sign((Quoted, pcr, x), inv(tpmk))
        TPM(pcr)
    } +
    receive Session, tpmk, aenc(esk, tpmk): {
        new sid;
        send sid;
        receive senc((Extend, x, sid), tpmk);
        send Extended, hash(x,pcr);
        TPM(hash(x, pcr))
    } +
    receive Decrypt, aenc(c, pk(k)), Created(k, pcr'): {
        if (pcr = pcr') then
         send dec(c, k)
        TPM(pcr)
    }
}
```

Unlike the projection from the choreography, the behaviour of the TPM API is just a sum of all the possible methods that can be invoked. Note that we also enhance the local behaviour specification with recursion. We conjecture that the projection of the TPM from the choreography and the API above can be formally related. Our idea is to look at the set of possible traces that the API can perform and compare to the traces of the projection (up-to recursive behaviour). Clearly, if a trace is in the projection of the choreography then it is for sure a trace of the API. This shows that the API implementation is compliant with the envelope protocol.

4 Verification in Tamarin

Our mechanised analysis is carried out in Tamarin [28], an interactive protocol verifier that can prove reachability and equivalence-based properties in the symbolic model. It has an expressive language based on multiset rewriting rules.

In Tamarin, terms are variables and functions ranging over terms; facts are predicates that store state information and are parameterized by terms; facts may be linear (i.e. can be consumed only once) or persistent (i.e. can be consumed arbitrarily often by rules); rules are essentially defined as transitions from one multiset of facts to another. The Tamarin multiset rewriting rules define a labeled transition system. The labels are used to reason about the behaviour of a protocol. Thus, to analyse the envelope protocol in Tamarin, we need to annotate our rules with appropriate labels that will serve to the specification of our

security properties. Tamarin encodes a Dolev-Yao [19] adversary that controls the network.

Conventionally, cryptographic primitives can be modelled in Tamarin by means of equational theories. An equational theory E describes the equations that hold on terms built from the signature. Terms are related by an equivalence relation $=$ induced by E. For instance, the equation $dec(enc(m, k), k) = m$ models a symmetric encryption scheme. The term m is the message, the term k is the secret key, the term *enc* models the encryption function, and the term *dec* models the decryption function, namely a deconstructor for the function *enc*.

Trace properties can be modelled in Tamarin via metric first-order logic. Predicates are labels and properties can be expressed using quantification over time. For example, the following lemma models a non-injective agreeement on the message x, meaning that for all got message x, there exists at least an event in which the message x has been previously sent.

$$\text{(Non-injective agreement)} \quad \forall x\ \#i.\ Get(x)@i \implies \exists\ \#j.\ Sent(x)@j \wedge j < i$$

The endpoint projection of the API of the TPM have immediate specifications into Tamarin rules: labels in a choreography can be translated to Tamarin's facts; `send` and `receive` can be mapped into the Tamarin's *Out* and *In* respectively; conditionals in choreography can be captured using pattern matching in Tamarin. For example, the rule below captures the `Decrypt` service.

```
rule Decrypt:
  let c = enc(v,~k) in
  [ In(c), In(Created(~k, pcr)), TPM(pcr) ]
   -[ Decrypt(v), TPM(pcr) ]->
  [ Out(v), TPM(pcr) ]
```

Here, the TPM outputs the value v, which is encrypted in c, if and only if the value of *pcr* in the TPM is equal to the value of *pcr* in `Created`.

Similarly, we can model Alice behaviour with two rules, one to capture Alice sending the commands Boot, Extend, and Create (*Alice_BEC*) and one to capture Alice sending the envelope (*Alice_Env*).

```
rule Alice_BEC:
  [ Fr(~n) ]
   -[ Alice_BEC(~n) ]->
  [ Out(Boot()), Out(Extend(~n)), Out(Create(hash('obt',
    hash(~n, 'nil')))),
    Alice1(~n) ]

rule Alice_Env:
  [ Alice1(~n), Fr(~v), In(Created(~k, hash('obt', hash(~n,
    'nil')))) ]
   -[ Alice_Env(~n, ~v, ~k) ]->
  [ Out(enc(~v, ~k)) ]
```

We do not need to model the role the parents since they act as an adversary, thus the role is controlled by the Dolev-Yao attacker encoded in Tamarin.

The specification of auditable transfer can be modelled in Tamarin as

$$\text{(Auditable transfer)} \quad \forall n\ v\ k\ \#i\ \#j.\ \mathit{Alice\_BEC}(n)@i \wedge \mathit{Alice\_Env}(n, v, k)@j \implies \neg(\exists\ \#l.\ !KU(v)@l) \vee \neg(\exists\ \#l.\ !KU(\mathit{Quoted}(h(\text{'ref'}, h(n, \text{'nil'})), \mathit{enc}(v, k))@l)$$

The fact *!KU* represents the knowledge of the adversary, which in our case is the parents. Thus, the property says that whenever Alice has initialised the TPM and sent the envelope, the adversary cannot both open the envelope and learn the content *and* obtain a proof of refusal of opening the envelope.

Tamarin cannot prove automatically auditable transfer. However, we resort to the interactive proof theory feature of Tamarin to find the proof. The proof and the full Tamarin code modelling the envelope protocol are available in [1].

5 Related Work

The idea of using choreographic languages for describing security protocol is not new: in fact, Alice and Bob notations are the predominant informal notation used by protocol designers.

The first work on a formal Alice and Bob notation with automated translation to the process algebra CSP is the Casper compiler by Lowe [27]. Mödersheim [30] later proposed an Alice and Bob-like language with a formal semantics and support for algebraic properties a-la Diffie-Hellman, that is integrated into the prover OFMC [5]. Several extension to this line of work have been made, for example, to support arbitrary algebraic reasoning [3] and secure and pseudonimous channels [33] forwarding channels [10]. Alice and Bob-style languages have a level of clarity and explainability that other models lack, however they only support linear protocols, which makes it impossible to represent API-like protocols such as the one analyzed in this paper, which typically contain branching and state that persists across sessions.

Stateful protocol verification is another relevant line of work. The first tool to support stateful verification was AIF [31], which abstracts values according to their membership class. This abstraction techniques was also applied to process algebras [9] and extended to countable families of sets [32] to support unbounded principals. The TPM envelope protocol that we use here as an example motivated the work on StatVerif [4], however the first analysis of the protocol was done with a custom encoding in Horn clauses [17]. Joshua and his colleagues also took the TPM envelope protocol as inspiration to extend the CPSA tool based on strand spaces to handle stateful protocols [22]. Another tool that supports stateful protocols is the Tamarin prover [28], which uses multiset rewrite rules to describe security protocols and we employ here as our target language. Later, support to stateful protocols was also added into the protocol verifier ProVerif [15].

Carbone and Guttmann [11] proposed a simple choreographic language with boxes for writing web interactions. Their core idea is that boxes containing information that must be exchanged in a network are annotated with the sender and

the receiver, respectively the creator of box and the one who can open it. Then, an endpoint projection is provided which generates local behaviour expressed in the Strand Spaces formalism. The translation introduces cryptographic enhancements to the boxes in order to ensure authentication and secrecy. Similarly, Bhargavan et al. [6] use a choreographic language inspired by multiparty session types for specifying web services. Similarly to Carbone and Guttmann, their tool adds some cryptographies to the messages specified in the choreography. Both research contributions differ from our idea in the fact that, unlike us, they abstract from the details that are necessary for achieving security properties. In particular, they do not consider expressing security properties at choreographic level.

The hallmark characteristic of auditable transfer is that it allows a party to get evidence on whether another party has learnt a secret. This seems an instantiation of the broader notion of *auditability* [21,24], which is defined as the quality of a protocol, which stores a sufficient number of pieces of evidence, to convince a third party that specific properties are satisfied. Similar properties are *verifiability*, which ensures that the failure of a protocol's goal can be detectable [16,25] and accountability [8,26], which additionally guarantees that misbehaving parties can be blamed. An interesting line of work is to study choreography for the modelling of such broader properties. For example, verifiability, accountability, and dispute resolution properties can all be defined by identifying the *tests* that decide whether a protocol's goal fails, and then check that each of the tests meets soundness, completeness, and sufficiency conditions [8,20,26]. Choreographies can be the language that enables the analyst to formulate tests and conditions, which can be then checked by a model checker of choice.

6 Conclusion

In this work we have given a formulation of the auditable transfer protocol as a choreography. We have shown how the role of the TPM can be defined with a local choreography, and that the auditable transfer protocol is shown to be a refinement of the TPM API. The choreographic constructs of branching and recursion useful language constructions to express the core properties of stateful, API-like protocols like the one we considered in this paper. It is then possible to translate to other intermediate languages for verification, like we have shown in our example for the multiset rewrite rules of Tamarin.

References

1. Tamarin code (2021). https://www.dropbox.com/sh/lonxu6vmj3iilmu/AAAErB3ATSNg59MFGxBcp74Ha?dl=0
2. Ables, K., Ryan, M.D.: Escrowed data and the digital envelope. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) Trust 2010. LNCS, vol. 6101, pp. 246–256. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13869-0_16

3. Almousa, O., Mödersheim, S., Viganò, L.: Alice and bob: reconciling formal models and implementation. In: Bodei, C., Ferrari, G.-L., Priami, C. (eds.) Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 66–85. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25527-9_7
4. Arapinis, M., Phillips, J., Ritter, E., Ryan, M.D.: Statverif: verification of stateful processes. J. Comput. Secur. **22**(5), 743–821 (2014). https://doi.org/10.3233/JCS-140501
5. Basin, D.A., Mödersheim, S., Viganò, L.: OFMC: a symbolic model checker for security protocols. Int. J. Inf. Sec. **4**(3), 181–208 (2005). https://doi.org/10.1007/s10207-004-0055-7
6. Bhargavan, K., Corin, R., Deniélou, P., Fournet, C., Leifer, J.J.: Cryptographic protocol synthesis and verification for multiparty sessions. In: Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, 8–10 July 2009, pp. 124–140. IEEE Computer Society (2009). https://doi.org/10.1109/CSF.2009.26
7. Brøndum, C.: Languages and Translators for Stateful Protocols. Tech. rep., DTU, MSc. Thesis (2020). https://findit.dtu.dk/en/catalog/2525864377
8. Bruni, A., Giustolisi, R., Schuermann, C.: Automated analysis of accountability. In: Nguyen, P., Zhou, J. (eds.) Information Security Conference, vol. 10599, pp. 417–434. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-69659-1_23
9. Bruni, A., Mödersheim, S., Nielson, F., Nielson, H.R.: Set-pi: Set membership p-calculus. In: Fournet, C., Hicks, M.W., Viganò, L. (eds.) IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13–17 July 2015, pp. 185–198. IEEE Computer Society (2015). https://doi.org/10.1109/CSF.2015.20
10. Bugliesi, M., Calzavara, S., Mödersheim, S., Modesti, P.: Security protocol specification and verification with anbx. J. Inf. Secur. Appl. **30**, 46–63 (2016). https://doi.org/10.1016/j.jisa.2016.05.004
11. Carbone, M., Guttman, J.D.: Choreographies with secure boxes and compromised principals. In: Bonchi, F., Grohmann, D., Spoletini, P., Tuosto, E. (eds.) Proceedings 2nd Interaction and Concurrency Experience: Structured Interactions, ICE 2009, Bologna, Italy, 31st August 2009. EPTCS, vol. 12, pp. 1–15 (2009). https://doi.org/10.4204/EPTCS.12.1
12. Carbone, M., Guttman, J.D.: Execution models for choreographies and cryptoprotocols. In: Beresford, A.R., Gay, S.J. (eds.) Proceedings Second International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES 2009, New York, UK, 22nd March 2009. EPTCS, vol. 17, pp. 31–41 (2009). https://doi.org/10.4204/EPTCS.17.3
13. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. **34**(2), 8:1–8:78 (2012). https://doi.org/10.1145/2220365.2220367
14. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Giacobazzi, R., Cousot, R. (eds.) The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy, 23–25 January 2013. pp. 263–274. ACM (2013). https://doi.org/10.1145/2429069.2429101
15. Cheval, V., Cortier, V., Turuani, M.: A little more conversation, a little less action, a lot more satisfaction: Global states in proverif. In: 31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, 9–12 July 2018, pp. 344–358. IEEE Computer Society (2018). https://doi.org/10.1109/CSF.2018.00032

16. Cortier, V., Galindo, D., Küsters, R., Müller, J., Truderung, T.: SoK: verifiability notions for e-voting protocols. In: IEEE Symposium on Security and Privacy, pp. 779–798 (2016)
17. Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on TPM state registers. In: Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27–29 June, 2011, pp. 66–80. IEEE Computer Society (2011). https://doi.org/10.1109/CSF.2011.12
18. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_41
19. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (1983)
20. Giustolisi, R., Bruni, A., et al.: Privacy-preserving dispute resolution in the improved bingo voting. In: Krimmer, R. (ed.) E-Vote-ID 2020. LNCS, vol. 12455, pp. 67–83. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60347-2_5
21. Guts, N., Fournet, C., Zappa Nardelli, F.: Reliable evidence: auditability by typing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 168–183. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_11
22. Guttman, J.D., Liskov, M.D., Ramsdell, J.D., Rowe, P.D.: Formal support for standardizing protocols with state. In: Chen, L., Matsuo, S. (eds.) SSR 2015. LNCS, vol. 9497, pp. 246–265. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27152-1_13
23. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. J. ACM **63**(1), 9:1–9:67 (2016). https://doi.org/10.1145/2827695
24. Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: Towards a theory of accountability and audit. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 152–167. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_10
25. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 389–404. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15497-3_24
26. Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: CCS, pp. 526–535. ACM (2010)
27. Lowe, G.: Casper: a compiler for the analysis of security protocols. J. Comput. Secur. **6**(1–2), 53–84 (1998). http://content.iospress.com/articles/journal-of-computer-security/jcs106
28. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
29. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes I and II. Inf. Comput. **100**(1), 1–77 (1992)
30. Mödersheim, S.: Algebraic properties in alice and bob notation. In: Proceedings of the The Forth International Conference on Availability, Reliability and Security, ARES 2009, 16–19 March 2009, Fukuoka, Japan, pp. 433–440. IEEE Computer Society (2009). https://doi.org/10.1109/ARES.2009.95

31. Mödersheim, S.: Abstraction by set-membership: verifying security protocols and web services with databases. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, 4–8 October 2010, pp. 351–360. ACM (2010). https://doi.org/10.1145/1866307.1866348
32. Mödersheim, S., Bruni, A.: AIF-ω: set-based protocol abstraction with countable families. In: Piessens, F., Viganò, L. (eds.) POST 2016. LNCS, vol. 9635, pp. 233–253. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49635-0_12
33. Mödersheim, S., Viganò, L.: Secure pseudonymous channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_21
34. W3C WS-CDL Working Group: Web services choreography description language version 1.0 (2004). http://www.w3.org/TR/ws-cdl-10/