Abstraction and Refinement in Protocol Derivation

Anupam Datta Ante Derek John C. Mitchell Dusko Pavlovic

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{danupam,aderek,jcm}@cs.stanford.edu

Kestrel Institute
Palo Alto, CA 94304
dusko@kestrel.edu

Abstract

Protocols may be derived from initial components by composition, refinement, and transformation. Adding function variables to a previous protocol logic, we develop an abstraction-instantiation method for reasoning about a class of protocol refinements. The main idea is to view changes in a protocol as a combination of finding a meaningful "protocol template" that contains function variables in messages, and producing the refined protocol as an instance of the template. Using higher-order protocol logic, we can develop a single proof for all instances of a template. A template can also be instantiated to another template, or a single protocol may be an instance of more than one template, allowing separate protocol properties to be proved modularly. These methods are illustrated using some challenge-response and key exchange protocol templates and an exploration of the design space surrounding JFK (Just Fast Keying) and related protocols from the IKE (Internet Key Exchange) family, which produces some interesting protocols not previously studied in the open literature.

1. Introduction

Many network protocols with security objectives are designed using a smaller set of common protocol concepts, such as challenge-response, Diffie-Hellman-like key agreement, and "cookies" to reduce potential denial of service attacks. In previous work [6, 7, 8], we proposed a protocol derivation framework, based on the use of composition, refinement, and transformation, and a formal logic for stating and proving properties of protocols. In this framework, a protocol designer may choose two initial protocol components, refine each of them, compose the results to get a candidate protocol, then apply one or more transformations to improve efficiency or resist particular forms of attack. While properties of the resulting protocol may be proved formally in our logic, the structure of protocol proofs we have previously devised have not always followed the structure of the protocol derivation. This paper extends the previous protocol logic with higher-order features, making it possible to define protocol templates and reason about their instances. Using protocol templates, we are able to characterize the correctness properties of a class of protocol refinements, furthering our long-term effort toward a framework for systematically deriving verified security protocols.

Composition combines separate protocols, refinements change the content or structure of individual messages, and transformations alter the structure of a protocol. In our formal logic, we may prove properties about a composed protocol from its parts, using a set of composition inference rules [7, 8]. The composition rules involve local reasoning about steps in each role and global reasoning about invariants in the protocol or set of protocols in use. In a protocol refinement, a message or portion of a message is systematically refined by, for example, adding additional data or otherwise changing the data contained in one or more messages. For example, replacing a plaintext nonce by an encrypted nonce, in both the sending and receiving protocol roles, is a protocol refinement. While refinements seem to arise naturally in contemporary practical protocols [2, 19], they provide more of a challenge for formal reasoning. One reason is that refinements may involve replacement, and replacement of one expression by another does not have a clean formulation in standard mathematical logic. This immediate problem is solved by introducing protocol templates and decomposing term replacement into an abstraction step of selecting an appropriate template and an instantiation step that replaces template variables with protocol expressions. Another issue, addressed by associating hy-

potheses with a proof about a template, is that a refinement may not apply to all protocols, but only to protocols that satisfy certain hypotheses.

To give a simple example, suppose we have a protocol containing messages that use symmetric encryption, and suppose that some useful property of this protocol is preserved if we replace symmetric encryption by use of a keyed hash. We can capture the relationship between these two protocols by writing an "abstract" protocol template with function variables in the positions occupied by either encryption or keyed hash. Then the two protocols of interest become instances of the template. In addition, a similar relationship often works out for protocol proofs. If we start with a proof of some property of the protocol that contains symmetric encryption, some branches of the proof tree will establish properties of symmetric encryption that are used in the proof. If we replace symmetric encryption by a function variable, then the protocol proof can be used to produce a proof about the protocol template containing function variables. This is accomplished by replacing each branch that proves a property of symmetric encryption by a corresponding hypothesis about the function variable. Once we have a proof for the protocol template obtained by abstracting away the specific uses of symmetric encryption, we can consider replacing the function variable with keyed hash. If keyed hash has the properties of symmetric encryption that were used in the initial proof, we can use proofs of these properties of keyed hash in place of the assumptions about the function variable. Thus an abstraction step and an instantiation step bring us both from a protocol with symmetric encryption to a protocol with keyed hash, and from a proof of the initial protocol to a proof of the final one. The role of the protocol template in this process is to provide a unified proof that leads from shared properties of two primitives (symmetric encryption or keyed hash) to a protocol property that holds with either primitive.

After describing the formal framework, we illustrate the use of protocol templates with several examples. As an example of multiple instantiations of a single template, we prove an authentication property of a generic challenge-response protocol, and then show how to instantiate the template to ISO-9798-2, ISO-9798-3, or SKID3 [26]. As an example of one protocol that is an instance of two templates, we show how to reason about an identity-protection refinement using an authentication template and an encryption template. The third example compares two key exchange protocol templates, one that can be instantiated to the ISO-9798 family of protocols, and one that can be instantiated to STS [10] and SIGMA [19]. The first reflects the key exchange mechanism used in JFKi [2], while the second corresponds to that of IKE [15], JFKr [2], and IKEv2 [17]. While there has been considerable debate and discussion in the IETF community about the tradeoffs offered by these two protocols, previous analyses are relatively low-level and do not illustrate the design principles involved. However, it is possible to compare the authentication and non-repudiation properties of the two approaches by comparing the templates.

While our past work on protocol derivation has given rational reconstructions of known protocols, we can also use protocol derivation to combine known protocols in new ways. We begin with two well-known protocol families: the first includes Diffie-Hellman key exchange and several variants that enhance the key derivation function (MTI/A [23], UM [3] and MQV [20]); the second is the IKE family – STS being the base protocol and a few steps further on is JFKr. These two protocol derivations provide a two-dimensional matrix of protocols that have not been explored, to our knowledge. The most sophisticated is a form of JFK, using MQV in place of Diffie-Hellman as its key-exchange component. This protocol provides forms of key secrecy, mutual authentication, forward secrecy, known-key security, computational efficiency, identity protection, and denial-of-service protection, inheriting these qualities from the parent derivations. The abstraction-instantiation method proves useful to reason formally about a subset of these protocols and associated security properties.

There are several differences between the work described in this paper and some other protocol analysis efforts. To begin with, our basic model of protocol execution and possible attacker actions is the traditional "Dolev-Yao model" [11, 29] that has been used in many other efforts [18, 28, 24, 30]. In particular, the protocol refinements we consider all replace symbolic operations of one form with symbolic operations of another; we do not consider "refining" a symbolic operation on message strings to a computable operation on bit sequences. While it is an important research direction to relate our model to computational models such as [5, 33, 27, 34], we currently believe that "computational soundness" of symbolic methods is a separable goal that will lead to greater use of the kind of logical methods considered in this paper. At a more detailed level, there are some important differences between the way that we reason about incremental protocol construction and alternative approaches such as "universal composability" [5]. In universal composability, properties of a protocol are stated in a strong form

so that the property will be preserved under a wide class of composition operations. In contrast, our protocol proofs proceed from various assumptions, including invariants that are assumed to hold in any environment in which the protocol operates. The ability to reason about protocol parts under assumptions about the way they will be used offers greater flexibility and appears essential for developing modular proofs about certain classes of protocols.

The rest of this paper is organized as follows. Section 2 reviews the framework developed in our previous work. The extension of the protocol logic with function variables is sketched in Section 2.4. Section 3 describes the abstraction-instantiation methodology which is the focus of this paper. Applications of this method to existing real-world security protocols are presented in Section 4. Section 5 explores the possibility of using this method for combining known protocols in new ways. Finally, Section 6 presents our conclusions and mentions some directions for future work.

2. Background

2.1. Derivation System

In [6], we outlined a protocol derivation framework and formalized a subset of it. The framework consists of a set of basic building blocks called *components* and a set of operations for constructing new protocols from old ones. These operations were divided into three general types: *composition*, *refinement*, and *transformation*. As mentioned in the introduction, composition combines separate protocols, refinement changes the content or structure of individual messages, and transformation alters the structure of a protocol. Some example components, composition operations, refinements and transformations were described but not fully formalized in [6]. Further examples and more comprehensive formalization of protocol composition appear in [8, 7].

So far, the protocol derivation framework consists of some informal protocol operations, a precise notation for defining protocols, a formal logic for proving properties of protocols, and some connections between the derivation operations and formal proofs. In subsections 2.2 and 2.3, we briefly review the protocol notation and formal logic. Subsection 2.4 contains a short explanation of how we extend both with concepts from higher-order logic in order to support the abstraction-instantiation method that is the focus of this paper.

2.2. Cord Calculus

One important part of security analysis involves understanding the way honest agents running a protocol will respond to messages from a malicious attacker. The common informal arrows-and-messages notation is therefore insufficient, since it only presents the intended executions (or traces) of the protocol. In addition, our protocol logic requires more information about a protocol than the set of protocol executions obtained from honest and malicious parties; we need a high-level description of the program executed by each agent performing each protocol role. As explained in [12], we used a form of process calculus that we call *cords*.

The STS protocol [10], written as a pair of cords, one for each role, is shown in Figure 1. The arrows between the cords are used in the figure to show how messages sent by one role may be received by the other, but they are not part of the cord formalism. The sequence of actions in the initiator role is given by the cord **A**. The notations (νx), $\langle t \rangle$, (x) refer respectively to the actions of nonce generation, sending a term, and receiving a message. A message is assumed to have the form: (source, destination, content). In words, the actions of **A** are: generate a fresh nonce; send a message using that number to $\hat{B}$; receive a message with source address $\hat{B}$; verify aspects of the message; and finally, send another message containing data received and the initial nonce generated at the start of the run.

2.3. A Protocol Logic

Our basic protocol logic and proof system are developed in [12, 6, 8, 7], with [8] providing a relatively succinct presentation of the most recent form.

The formulas of the logic are given by the grammar in Table 1, where ρ may be any role, written using the notation of cord calculus. Here, t and P denote a term and a process respectively. We use the word *process* to refer to a principal executing an instance of a role. As a notational convention, we use X to refer to a process belonging to principal $\hat{X}$. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message".

Most protocol proofs use formulas of the form $\theta[P]_X\phi$, which means that after actions P are executed in process X, starting from a state where formula θ is true, formula ϕ is true about the resulting state of X. Here are the informal interpretations of the predicates (see [8] for detailed semantics):

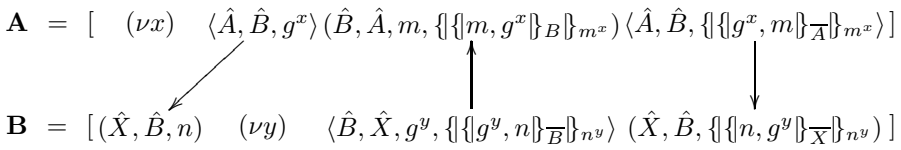

Figure 1. STS as a cord space

$\mathsf{Has}(X, x)$ means principal $\hat{X}$ possesses information x in the process X. This is "possess" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known.

$\mathsf{Send}(X, m)$ means principal $\hat{X}$ sends message m in the process X.

$\mathsf{Receive}(X, m)$, $\mathsf{New}(X, t)$, $\mathsf{Decrypt}(X, t)$, $\mathsf{Verify}(X, t)$ similarly mean that receive, new, decrypt and signature verification actions occur.

$\mathsf{Fresh}(X, t)$ means the term t generated in X is "fresh" in the sense that no one else has seen any term containing t as a subterm. Typically, a fresh term will be a nonce and freshness will be used to reason about the temporal ordering of actions in runs of a protocol.

$\mathsf{Honest}(\hat{X})$ means the actions of principal $\hat{X}$ in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, $\hat{X}$ assumes some set of roles and does exactly the actions prescribed by them.

$\mathsf{Computes}(X, m)$ means that principal $\hat{X}$ possesses enough information in process X to build term m of certain forms. For example, $\mathsf{Computes}(X, \{|t|\}_K)$ holds if X possesses both the term t and the key K (see Appendix A for further details).

$\mathsf{Contains}(t_1, t_2)$ means that t_2 is a subterm of t_1.

The two temporal operators $\diamondminus$ and $\ominus$ have the same meaning as in Linear Temporal Logic [22]. Since we view a run as a linear sequence of states, $\diamondminus\, \phi$ means that in some state in the past ϕ holds, whereas $\ominus\, \phi$ means that in the previous state ϕ holds.

The predicate $\mathsf{After}(\mathsf{a}_1, \mathsf{a}_2)$, definable from $\diamondminus$ and $\ominus$, means that the action a_2 happened after the action a_1 in a run.

2.4. Cords and Protocol Logic with Function Variables

Like a program module containing functions that are not defined in the module, a cord may contain functions that are not given a specific meaning in the cord calculus. When a cord contains undefined functions, the cord cannot be executed as is, but can be used to define a set of runs if the function is replaced by a combination of defined operations. Since cords contain functions such as encryption and pairing, it is a simple matter to extend the syntax with additional function names. Since we will apply substitution for these function names, and implicitly quantify over their possible interpretations in the protocol logic, we refer to these function names as function variables. The mechanism for substituting an expression for a function variable, in a manner that treats function arguments correctly, is standard in higher-order logic. A simple explanation that does not involve lambda calculus or related machinery is given at the beginning of [14].

In a judgement

$$\mathcal{Q}, \Gamma \vdash \phi_1 [P]_A \phi_2$$

where $\mathcal{Q}$ is a protocol containing function variables, P is one role or initial segment of a role of the protocol, and Γ denotes the set of assumed properties and invariants. The formulas in Γ may also contain function variables. The meaning of this judgement is that for every substitution that eliminates all function variables, any execution of the resulting protocol $\mathcal{Q}'$ respecting the resulting invariants Γ' satisfies the resulting formula $\phi_1' [P']_A \phi_2'$.

Theorem 2.1 (Soundness Theorem) *Security Protocol Logic [12, 6, 13, 8] is sound for protocols and assertions containing function variables. Furthermore, substitution preserves semantic entailment and validity of formulas.*

As a technical note for logicians, we observe that since we do not have any comprehension principle for our logic, it is actually reducible to first-order logic by the standard method of treating function variables as first-order variables via an Apply function. Consequently, our higher-order protocol logic is no less tractable for automated theorem proving than the logic without function variables.

Action formulas
$\mathsf{a} \quad ::= \quad \mathsf{Send}(P,m) \mid \mathsf{Receive}(P,m) \mid \mathsf{New}(P,t) \mid \mathsf{Decrypt}(P,t) \mid \mathsf{Verify}(P,t)$
Formulas
$\phi \quad ::= \quad \mathsf{a} \mid \mathsf{Has}(P,t) \mid \mathsf{Computes}(P,t) \mid \mathsf{Fresh}(P,t) \mid \mathsf{Honest}(N) \mid \mathsf{Contains}(t_1,t_2) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi \mid \diamondminus \phi \mid \ominus \phi$
Modal forms
$\Psi \quad ::= \quad \rho\ \phi \mid \phi\ \rho\ \phi$

Table 1. Syntax of the logic

3. Abstraction and Refinement Methodology

Protocol Templates: A *protocol template* is a protocol that uses function variables. An example of a challenge-response protocol template using the informal trace notation is given below.

$$\begin{array}{l} A \rightarrow B : m \\ B \rightarrow A : n, F\,(B, A, n, m) \\ A \rightarrow B : G\,(A, B, m, n) \end{array}$$

Here, m and n are fresh nonces and F and G are function variables. Substituting cryptographic functions for F and G with the parameters appropriately filled in yields concrete protocols. For example, instantiating F and G to signatures yields the standard signature-based challenge-response protocol from the ISO-9798-3 family, whereas instantiating F and G to a keyed hash yields the SKID3 protocol.

Characterizing protocol concepts: Protocol templates provide a useful method for formally characterizing design concepts. Our methodology for formal proofs involves the following two steps.

1. Assuming properties of the function variables and some invariants, prove properties of the protocol templates. Formally,

   $$\mathcal{Q}, \Gamma \vdash \phi_1 [P]_A \phi_2$$

   Here, $\mathcal{Q}$ is an abstract protocol and P is a program for one role of the protocol. Γ denotes the set of assumed properties and invariants.

2. Instantiate the function variables to cryptographic functions and prove that the assumed properties and invariants are satisfied by the obtained protocol. Hence conclude that this protocol possesses the security property characterized by the protocol template.

   $$\text{If } \mathcal{Q}' \vdash \Gamma', \text{ then } \mathcal{Q}' \vdash \phi_1' [P']_A \phi_2'$$

   Here, the primed versions of the protocol, hypotheses, etc. are obtained by applying the substitution σ used in the instantiation.

The correctness of the method is an immediate corollary of Theorem 2.1.

Combining protocol templates: Protocol templates can also be used to formalize the informal practice of protocol design by combining different mechanisms. The key observation is that if a concrete protocol is an instantiation of two different protocol templates, each instantiation respecting the assumed invariants associated with the template, then the concrete protocol has the security properties of both templates. Our methodology involves the following three steps.

1. Identify two protocol templates which guarantee certain security properties under some assumptions.

   $$\mathcal{Q}_1, \Gamma_1 \vdash \phi_{11} [P_1]_A \phi_{21} \text{ and } \mathcal{Q}_2, \Gamma_2 \vdash \phi_{12} [P_2]_A \phi_{22}$$

   Here, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are protocol templates; P_1 and P_2 are respectively the programs corresponding to a specific role; and Γ_1 and Γ_2 denote the sets of assumed properties and invariants.

2. Find substitutions σ_1 and σ_2 such that the two instantiated protocols and roles are identical, i.e.,

   $$\sigma_1 \mathcal{Q}_1 = \sigma_2 \mathcal{Q}_2 = \mathcal{Q}' \text{ and } \sigma_1 P_1 = \sigma_2 P_2 = P'$$

3. Prove that the instantiated protocol satisfies the hypotheses of both the protocol templates. Hence conclude that it inherits the security properties of both.

   $$\text{If } \mathcal{Q}' \vdash \Gamma_1' \cup \Gamma_2', \text{then } \mathcal{Q}' \vdash (\phi_{11}' \wedge \phi_{12}') [P']_A (\phi_{21}' \wedge \phi_{22}')$$

   Here, the primed versions of the protocol, hypotheses, etc. are obtained by applying the substitutions σ_1 and σ_2 used in the instantiations.

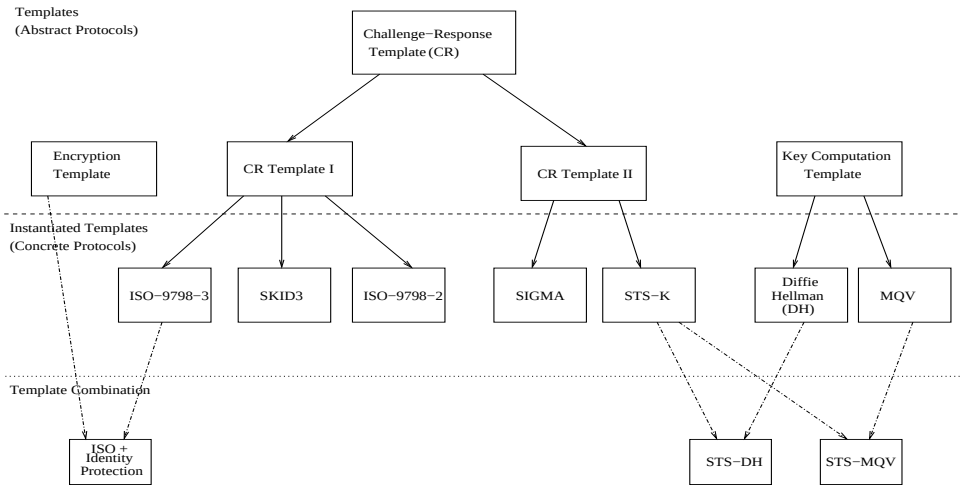

Figure 2. Illustrating the Methodology

4. Illustrative Examples

In this section, we present several examples illustrating the abstraction-instantiation methodology. The protocols considered include real-world protocols from the ISO and IKE families.

4.1. Characterizing Protocol Concepts

A protocol template can be instantiated to multiple protocols. Security proofs of instances of a template follow from the proof of the template plus a (usually much simpler) proof that the instances satisfy the assumed hypotheses. In what follows, we work through an example demonstrating the approach.

4.1.1. Example: Challenge-Response Template In our first example, we characterize a challenge-response protocol template and then obtain three protocols: ISO-9798-2, ISO-9798-3, and SKID3 by appropriate substitutions. In doing so, we follow the two step methodology outlined in Section 3.

Step 1*:* The first step is to precisely define and characterize the protocol template. This involves defining the template (denoted Q_{CR} in the sequel) as a cord space, expressing the security property achieved, and identifying the set of assumptions under which the property holds. The programs for the initiator and responder roles of Q_{CR} is written out below in the notation of cords.

$$\begin{aligned}\mathbf{Init_{CR}} \;=\; & (\nu m)\,\langle\{\hat{A},\hat{B},m\}\rangle \\ & (\{\hat{B},\hat{A},n,F(\hat{B},\hat{A},n,m)\}) \\ & \langle\{\hat{A},\hat{B},G(\hat{A},\hat{B},m,n)\}\rangle\end{aligned}$$

$$\begin{aligned}\mathbf{Resp_{CR}} \;=\; & (\{\hat{Y},\hat{B},y\}) \\ & (\nu x)\,\langle\{\hat{B},\hat{Y},x,F(\hat{B},\hat{Y},x,y)\}\rangle \\ & (\{\hat{Y},\hat{B},G(\hat{Y},\hat{B},y,x)\})\end{aligned}$$

Here, F and G are function variables. Under a set of assumptions (Γ_{CR}) about these variables, we prove an authentication property for the initiator of the protocol using the logic (see Table 2 for the complete formal proof).

$$Q_{CR},\Gamma_{CR} \;\vdash\; [\mathbf{Init_{CR}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \phi_{auth}$$

Intuitively, this formula means that if A executed a session of Q_{CR} supposedly with B and both of them are honest (implying that they strictly follow the protocol and do not, for example, reveal their private keys), then the authentication property expressed by the formula ϕ_{auth} holds in the resulting state. ϕ_{auth} specifies an authentication property for the initiator based on the concept of *matching conversations* [10]. Simply put, it requires that whenever A completes a session supposedly with B, both A and B have consistent views of the run, i.e., they agree on the content and order of the messages exchanged. Formally,

$$\begin{aligned}\phi_{auth} \;\equiv\; & \exists B.(\mathsf{ActionsInOrder}(\\ & \quad \mathsf{Send}(A,\{\hat{A},\hat{B},m\}), \\ & \quad \mathsf{Receive}(B,\{\hat{A},\hat{B},m\}), \\ & \quad \mathsf{Send}(B,\{\hat{B},\hat{A},n,F(\hat{B},\hat{A},n,m)\}), \\ & \quad \mathsf{Receive}(A,\{\hat{B},\hat{A},n,F(\hat{B},\hat{A},n,m)\}), \\ & \quad \mathsf{Send}(A,\{\hat{A},\hat{B},G(\hat{A},\hat{B},m,n)\}), \\ & \quad \mathsf{Receive}(B,\{\hat{A},\hat{B},G(\hat{A},\hat{B},m,n)\})))\end{aligned}$$

The set of assumptions Γ_{CR} used to prove the authentication property consists of the following four logical formulas:

$$\begin{aligned}
\gamma_1 \equiv\ & \mathsf{Computes}(X, F(\hat{B}, \hat{A}, n, m)) \supset \\
& (\exists A'.X = A') \vee (\exists B'.X = B') \\
\gamma_2 \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \supset \\
& \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, x\}, F(\hat{B}, \hat{A}, n, m)) \\
\gamma_3 \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \wedge \diamondminus\mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, G(\hat{X}, \hat{Y}, x, y)\}, F(\hat{B}, \hat{A}, n, m)) \supset \\
& \hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge n = y \wedge m = x \\
\gamma_4 \equiv\ & \diamondminus\mathsf{Fresh}(Z, x) \wedge \diamondminus\mathsf{Fresh}(W, y) \wedge \\
& \mathsf{Contains}(\{\hat{X}, \hat{Y}, x, F(\hat{X}, \hat{Y}, x, y)\}, F(\hat{B}, \hat{A}, n, m)) \supset \\
& \hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge n = x \wedge m = y
\end{aligned}$$

Informally, assumption γ_1 states that the function F is hard to compute: only agents $\hat{A}$ and $\hat{B}$ can compute $F(\hat{B}, \hat{A}, n, m)$ (more precisely, if some session X has enough information to compute $F(\hat{B}, \hat{A}, n, m)$ then X is either a session of agent $\hat{A}$ or agent $\hat{B}$). The honesty of $\hat{A}$ and $\hat{B}$ is a part of the premise and have been omitted to improve readability. In a concrete protocol, this assumption can be satisfied by, for example, instantiating F to a signature. The other assumptions impose syntactic constraints on F and G. For example, γ_2 implies that $F(\hat{B}, \hat{A}, n, m)$ cannot be mistaken for a nonce. This obviates certain type confusion attacks. γ_4 implies that F depends on the value of all four parameters.

Proof Structure of Challenge-Response Template: A complete proof of the authentication property for the initiator role of the challenge-response template is given in Table 2. The leftmost column identifies the axioms and rules used in the corresponding step, where the naming convention follows [8] and Appendix A. The proof naturally breaks down into four parts:

- Line (1) asserts what actions were executed by Alice in the initiator role. Specifically, we can conclude that Alice has received a message msg containing $F(\hat{B}, \hat{A}, n, m)$.
- In lines (2)–(9), we track the source of term $F(\hat{B}, \hat{A}, n, m)$ received by Alice. Since Alice received a message containing $F(\hat{B}, \hat{A}, n, m)$, there must be a process which computed that term and sent it out. Using the assumption γ_1, we can conclude that only Alice or Bob could have computed $F(\hat{B}, \hat{A}, n, m)$. From assumptions $\gamma_2, \gamma_3, \gamma_4$, we can deduce that Alice did not send $F(\hat{B}, \hat{A}, n, m)$. Therefore, Bob must have sent a message msg' containing $F(\hat{B}, \hat{A}, n, m)$.
- In lines (10)–(13), we use the honesty rule, and assumptions $\gamma_2, \gamma_3, \gamma_4$ to conclude that Bob must have sent $F(\hat{B}, \hat{A}, n, m)$ as part of the second message of the responder role. Therefore, Bob must have received a corresponding first message in the past. Also, using γ_4, we can conclude that Bob is in a session with Alice.
- Finally, in lines (14)–(19), the temporal ordering rules are used to establish a total ordering among the send-receive actions of Alice and Bob. Line (17) concludes that Bob must have received $msg1$ after Alice sent it since $msg1$ contains a fresh nonce. Line (18) uses the same argument for $msg2$ sent by Bob. Finally, line (19) uses the transitivity axiom to conclude that the authentication formula ϕ_{auth} is true.

This completes the characterization of the protocol template. We are now ready to move on to Step 2.

Step 2: In this step, we instantiate the protocol template to three well known protocols from the ISO family. The substitutions for the function variables and the resulting protocols are shown in Figure 3. ISO 9798-2, SKID3, and ISO 9798-3 [26] respectively use symmetric key encryption with a pre-shared key, keyed hash and signatures to instantiate F and G. These substitutions respect the assumed invariants in Γ_{CR}. For example, γ_1 is satisfied by signatures since the signature can be computed only by an agent who has the corresponding private key. (The formal proofs follow immediately from logical axioms and are omitted.) We can therefore conclude that all three protocols guarantee the authentication property characterized by the protocol template.

Note that we have only proved the authentication property for the initiator in the protocol. To complete the proof of the mutual authentication property, we need to prove a formula analogous to ϕ_{auth} for the responder. This is achieved using symmetric assumptions about function variable G. All three instantiations satisfy these additional assumptions.

4.2. Combining Protocol Templates

A refinement operation, when applied to a protocol, adds an additional security property while preserving the original properties. Examples of refinement operations considered in [6] include replacing signatures by encrypted signatures to provide identity protection and replacing fresh Diffie-Hellman exponentials by a pair consisting of a stale exponential and a fresh nonce, thereby enabling reuse of exponentials and hence greater computational efficiency. The methodology for combining protocol templates, described in Section 3, provides a way to formally reason about a broad

$\mathbf{AA1}, \mathbf{T1}, \mathbf{P1}$	$[\mathbf{Init_{CR}}]_A \diamondsuit \mathsf{Receive}(A, msg) \wedge \mathsf{Contains}(msg, F(\hat{B}, \hat{A}, n, m))$	(1)
$\mathbf{CP3}, (1)$	$[\mathbf{Init_{CR}}]_A \exists X. \exists msg'. (\mathsf{Computes}(X, F(\hat{B}, \hat{A}, n, m)) \wedge$ $\diamondsuit \mathsf{Send}(X, msg') \wedge \mathsf{Contains}(msg', F(\hat{B}, \hat{A}, n, m)))$	(2)
Γ_{CR}	$\mathsf{Computes}(X, F(\hat{B}, \hat{A}, n, m)) \supset (\exists A'. X = A') \vee (\exists B'. X = B')$	(3)
$\mathbf{HON}$	$\mathsf{Honest}(\hat{Y}) \wedge \diamondsuit \mathsf{Send}(Y, msg') \supset \exists X. \exists x. \exists y. (\diamondsuit \mathsf{Fresh}(Y, y) \wedge$ $(msg' = \{\hat{Y}, \hat{X}, y\} \vee$ $msg' = \{\hat{Y}, \hat{X}, y, F(\hat{Y}, \hat{X}, y, x)\} \vee$ $msg' = \{\hat{Y}, \hat{X}, G(\hat{Y}, \hat{X}, y, x)\})$	(4)
Γ_{CR}	$\neg \mathsf{Contains}(\{\hat{A}, \hat{X}, m'\}, F(\hat{B}, \hat{A}, n, m))$	(5)
Γ_{CR}	$\mathsf{Contains}(\{\hat{A}, \hat{X}, G(\hat{A}, \hat{X}, m', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset \hat{A} = \hat{B}$	(6)
Γ_{CR}	$\mathsf{Contains}(\{\hat{A}, \hat{X}, m', F(\hat{A}, \hat{X}, m', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset \hat{A} = \hat{B}$	(7)
$(4-7)$	$\mathsf{Honest}(\hat{A}) \wedge \diamondsuit \mathsf{Send}(A', msg') \wedge \mathsf{Contains}(msg', F(\hat{B}, \hat{A}, n, m)) \supset$ $\hat{A} = \hat{B}$	(8)
$(2-3), (8)$	$[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{A}) \supset \exists B. \exists msg'. (\mathsf{Computes}(B, F(\hat{B}, \hat{A}, n, m)) \wedge$ $\diamondsuit \mathsf{Send}(B, msg') \wedge \mathsf{Contains}(msg', F(\hat{B}, \hat{A}, n, m)))$	(9)
Γ_{CR}	$\neg \mathsf{Contains}(\{\hat{B}, \hat{X}, n'\}, F(\hat{B}, \hat{A}, n, m))$	(10)
Γ_{CR}	$\mathsf{Contains}(\{\hat{B}, \hat{X}, G(\hat{B}, \hat{X}, n', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset m = n'$	(11)
Γ_{CR}	$\mathsf{Contains}(\{\hat{B}, \hat{X}, n', F(\hat{B}, \hat{X}, n', x)\}, F(\hat{B}, \hat{A}, n, m)) \supset$ $\hat{A} = \hat{B} \wedge n = n' \wedge x = m$	(12)
$(4), (9-12)$	$[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \exists B.$ $(\diamondsuit \mathsf{Send}(B, \{\hat{B}, \hat{X}, G(\hat{B}, \hat{X}, n', x)\}) \wedge \diamondsuit \mathsf{Fresh}(B, n') \wedge n' = m) \vee$ $(\diamondsuit \mathsf{Send}(B, \{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\}))$	(13)
$\mathbf{AN3}, \mathbf{P1}$	$[\mathbf{Init_{CR}}]_A \diamondsuit \mathsf{Fresh}(A, m)$	(14)
$(13-14)$	$[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset$ $\diamondsuit \mathsf{Send}(B, \{\hat{B}, \hat{A}, n, F(\hat{B}, \hat{A}, n, m)\})$	(15)
$(15), \mathbf{HON}$	$[\mathbf{Init_{CR}}]_A \mathsf{Honest}(A) \wedge \mathsf{Honest}(\hat{B}) \supset \mathsf{ActionsInOrder}($ $\mathsf{Receive}(B, \{\hat{A}, \hat{B}, n\}), \mathsf{Send}(B, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}))$	(16)
$\mathbf{F}, \mathbf{AF3}$	$[\mathbf{Init_{CR}}]_A \mathsf{After}(\mathsf{Send}(A, \{\hat{A}, \hat{B}, n\}), \mathsf{Receive}(B, \{\hat{A}, \hat{B}, n\}))$	(17)
$\mathbf{F}, \mathbf{AF3}, \mathbf{HON}$	$[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{B}) \supset \mathsf{After}(\mathsf{Send}(B, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}),$ $\mathsf{Receive}(A, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}))$	(18)
$\mathbf{AF1}, \mathbf{AF2}$	$[\mathbf{Init_{CR}}]_A \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \phi_{auth}$	(19)

Table 2. Deductions of $\hat{A}$ executing $\mathbf{Init_{CR}}$ role

class of refinements including the two just mentioned. Below we illustrate the general method by examining the identity protection refinement in some detail.

4.2.1. Example: Identity Protection Refinement In this example, we start with a signature based protocol, ISO-9798-3, that provides mutual authentication. We apply the identity protection refinement to it, which involves replacing the signatures by encrypted signatures using a shared key. The intention is to prevent adversaries from observing signatures since they can reveal identities of communicating peers. Our goal is to prove that this refinement step is correct, i.e., it does indeed guarantee that the resulting protocol provides identity protection, while preserving the mutual authentication property of the original protocol. We identify two templates: Q_{CR}, the challenge-response template described in the previous section, and Q_{ENC} described below, which provides a form of secrecy. The aim now is to prove that the protocol obtained after the refinement step is an invariant respecting instance of both these templates and the terms protected by the secrecy template are precisely the signatures.

Step 1*:* The Q_{CR} template has been defined and characterized in an earlier section. Here, we do the same for the Q_{ENC} template. Using the informal arrows-and-messages diagram, the template can be described as

$F(X,Y,x,y) \equiv E_{K_{XY}}(x,y,X)$ $G(X,Y,x,y) \equiv E_{K_{XY}}(y,x)$	$F(X,Y,x,y) \equiv H_{K_{XY}}(x,y,X)$ $G(X,Y,x,y) \equiv H_{K_{XY}}(y,x,X)$	$F(X,Y,x,y) \equiv SIG_X(x,y,Y)$ $G(X,Y,x,y) \equiv SIG_X(y,x,Y)$
$A \to B : m$ $B \to A : n, E_{K_{AB}}(n,m,B)$ $A \to B : E_{K_{AB}}(n,m)$	$A \to B : m$ $B \to A : n, H_{K_{AB}}(n,m,B)$ $A \to B : H_{K_{AB}}(n,m,A)$	$A \to B : m$ $B \to A : n, SIG_B(n,m,A)$ $A \to B : SIG_A(n,m,B)$
ISO 9798-2	SKID3	ISO 9798-3

Figure 3. Instantiations of the Challenge-Response template

follows.

$$\begin{array}{l} A \to B : m \\ B \to A : n, E_{K_{AB}}(H(B,A,n,m)) \\ A \to B : E_{K_{AB}}(I(A,B,m,n)) \end{array}$$

The programs for the initiator and responder roles of Q_{ENC} is written out below in the notation of cords.

$$\begin{array}{rcl} \mathbf{Init_{ENC}} & = & (\nu m)\,\langle\{\hat{A},\hat{B},m\}\rangle \\ & & (\{\hat{B},\hat{A},n,E_{K_{AB}}(H(\hat{B},\hat{A},n,m))\}) \\ & & \langle\{\hat{A},\hat{B},E_{K_{AB}}(I(\hat{A},\hat{B},m,n))\}\rangle \end{array}$$

$$\begin{array}{rcl} \mathbf{Resp_{ENC}} & = & (\{\hat{Y},\hat{B},y\}) \\ & & (\nu x)\,\langle\{\hat{B},\hat{Y},x,E_{K_{BY}}(H(\hat{B},\hat{Y},x,y))\}\rangle \\ & & (\{\hat{Y},\hat{B},E_{K_{BY}}(I(\hat{Y},\hat{B},y,x))\}) \end{array}$$

Here, H and I are function variables. Under a set of assumptions (Γ_{ENC}) about these variables, we prove that the term $H(\hat{B},\hat{A},n,m)$ remains secret: it is known only to A and B. Formally,

$$Q_{ENC}, \Gamma_{ENC} \vdash [\mathbf{Init_{ENC}}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \phi_{secret}$$

Intuitively, this formula means that if A executed a session of Q_{ENC} supposedly with B and both of them are honest, then the secrecy property expressed by the formula ϕ_{secret} holds in the resulting state. ϕ_{secret} specifies the secrecy property for the term $H(\hat{B},\hat{A},n,m)$. Formally,

$$\phi_{secret} \equiv \exists B.(\mathsf{Has}(X, H(\hat{B},\hat{A},n,m)) \supset (X = A \vee X = B))$$

The set of assumptions Γ_{ENC} used to prove the authentication property consists of the following four logical formulas:

$$\begin{array}{l} \delta_1 \equiv \mathsf{Computes}(X, H(\hat{B},\hat{A},n,m)) \supset \exists B.X = B \\ \delta_2 \equiv \diamondsuit\mathsf{Fresh}(Z,x) \supset \\ \quad \neg\mathsf{Contains}(\{\hat{X},\hat{Y},x\}, H(\hat{B},\hat{A},n,m)) \\ \delta_3 \equiv \diamondsuit\mathsf{Fresh}(Z,x) \wedge \diamondsuit\mathsf{Fresh}(W,y) \wedge \\ \quad \mathsf{Contains}(\{\hat{X},\hat{Y},E_{K_{XY}}(I(\hat{X},\hat{Y},x,y))\}, H(\hat{B},\hat{A},n,m)) \\ \quad \supset \quad (\hat{X} = \hat{B} \wedge n = y \wedge m = x) \\ \delta_4 \equiv \diamondsuit\mathsf{Fresh}(Z,x) \wedge \diamondsuit\mathsf{Fresh}(W,y) \wedge \\ \quad \mathsf{Contains}(\{\hat{X},\hat{Y},x,E_{K_{XY}}(H(\hat{X},\hat{Y},x,y))\}, \\ \qquad H(\hat{B},\hat{A},n,m)) \supset \quad (\hat{X} = \hat{B} \wedge n = x \wedge m = y) \end{array}$$

These formulas capture simple ideas, e.g., $H(\hat{B},\hat{A},n,m)$ (e.g. $B's$ signature) can be computed only by B and certain syntactic constraints, e.g., a signature is not a subterm of a nonce.

Step 2*:* The second step is to find substitutions σ_1 and σ_2 such that both the templates (Q_{CR} and Q_{ENC}) instantiate to the same real protocol. The desired substitutions are shown in Figure 4. σ_1 is on the left, σ_2 is on the right and the instantiated protocol is in the middle of the figure.

Step 3*:* The final step is to verify that the instantiated protocol satisfies the union of the hypotheses in Γ_{CR} and Γ_{ENC}. This follows easily from the properties of signature and encryption under symmetric key as expressed in the logic and the syntactic structure of the protocol. We can therefore conclude that the identity protection refinement operation as applied here is correct, i.e., it adds the identity protection property while preserving the original properties of the protocol (which in this case is mutual authentication).

4.3. Authenticated Key-Exchange Templates

An important use of protocol templates is to underpin basic principles used in designing classes of protocols and to bring out subtle tradeoffs offered by various protocol families. In this section, we examine two families of authenticated key exchange protocols. The first template, AKE1, generalizes a family of protocols in which authentication is achieved by explicitly embedding the intended recipient's identity inside authenticators in messages. This family includes the ISO-9798-3 key exchange protocol and related protocols including the core JFKi protocol. The second template, AKE2, generalizes a family of protocols where agents authenticate each other using a combination of signatures and a proof of possession of the Diffie-Hellman shared secret computed during the execution of the protocol. This family includes STS, SIGMA, and the core of the IKE and JFKr protocols. Part of the reason these two

$F(X,Y,x,y) \equiv E_{K_{XY}}(SIG_X(x,y))$ $\quad A \rightarrow B : m$ $\quad H(X,Y,x,y) \equiv SIG_X(x,y)$
$G(X,Y,x,y) \equiv E_{K_{XY}}(SIG_X(y,x))$ $\quad B \rightarrow A : n, E_{K_{AB}}(SIG_B(n,m))$ $\quad I(X,Y,x,y) \equiv SIG_X(y,x)$
$A \rightarrow B : E_{K_{AB}}(SIG_A(n,m))$

Figure 4. Protocol that is an instantiation of both CR and ENC templates

families are interesting is that they were both candidates for the recently proposed IKEv2 protocol and there has been considerable discussion and debate in the IETF community about the tradeoffs offered by the two designs. The use of templates to characterize the two families sheds light on the subtle difference between the authentication, non-repudiation and identity protection guarantees associated with the two sets of protocols.

Template AKE1: Using the informal arrows-and-messages diagram, the authenticated key-exchange template AKE1 can be described as follows.

$$A \rightarrow B : A, g^a$$
$$B \rightarrow A : g^b, F(B, A, g^b, g^a)$$
$$A \rightarrow B : G(A, B, g^a, g^b)$$

For this template, we are able to prove both secrecy and authentication for the initiator role:

$$Q_{AKE1}, \Gamma_{AKE1} \vdash [\mathbf{Init_{AKE1}}]_A \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(B) \supset \phi_{auth} \wedge \phi_{shared-secret}$$

The formula ϕ_{auth} describes an authentication property for the initiator based on matching conversations, while formula $\phi_{shared-secret}$ states that A and B are the only two sessions which know the Diffie-Hellman secret g^{ab}. The set of assumptions Γ_{AKE1} is similar to Γ_{CR} in Section 4.1.1:

$$\epsilon_1 \equiv \mathsf{Computes}(X, F(\hat{B}, \hat{A}, g^b, g^a)) \supset \exists B'.X = B'$$
$$\epsilon_2 \equiv \diamondminus\mathsf{Fresh}(Z,x) \supset \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, g^x\}, F(\hat{B}, \hat{A}, g^b, g^a))$$
$$\epsilon_3 \equiv \diamondminus\mathsf{Fresh}(Z,x) \wedge \diamondminus\mathsf{Fresh}(W,y) \wedge \mathsf{Contains}(\{\hat{X}, \hat{Y}, G(\hat{X}, \hat{Y}, g^x, g^y)\}, F(\hat{B}, \hat{A}, g^b, g^a)) \supset (\hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge g^b = g^y \wedge g^a = g^x)$$
$$\epsilon_4 \equiv \diamondminus\mathsf{Fresh}(Z,x) \wedge \diamondminus\mathsf{Fresh}(W,y) \wedge \mathsf{Contains}(\{\hat{X}, \hat{Y}, g^y, F(\hat{X}, \hat{Y}, g^x, g^y)\}, F(\hat{B}, \hat{A}, g^b, g^a)) \supset (\hat{X} = \hat{B} \wedge \hat{Y} = \hat{A} \wedge g^b = g^x \wedge g^a = g^y)$$

We prove that the ISO-9798-3 key exchange protocol satisfies the set of assumptions, Γ_{AKE1}, and therefore provides similar authentication and secrecy guarantees. However, STS, SIGMA and their variants do not satisfy Γ_{AKE1}. Specifically, the assumption ϵ_4 fails since the intended recipient's identity is not embedded inside an authenticator in the second message of the protocol. The way the proof fails leads us to a run which provides a counterexample to the strong authentication property. This works for both STS and SIGMA and the run is essentially similar to the "attack" on STS first demonstrated by Lowe in [21].

Template AKE2: Using the informal arrows-and-messages diagram, AKE2 can be described as follows.

$$A \rightarrow B : g^a$$
$$B \rightarrow A : g^b, F(B, g^b, g^a), F'(B, g^{ab})$$
$$A \rightarrow B : G(A, g^a, g^b), G'(A, g^{ab})$$

Informally, the purpose of F is to ensure that B is a session with parameters g^a and g^b, while F' proves that B has the shared secret g^{ab}. More precisely, using the set of assumptions about the function variables Γ_{AKE2}, it is possible to prove that this protocol template provides a form of authentication – matching conversations for the responder:

$$Q_{AKE2}, \Gamma_{AKE2} \vdash [\mathbf{Init_{AKE2}}]_B \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \supset \phi_{auth} \wedge \phi_{shared-secret}$$

The set of assumptions Γ_{AKE2} is:

$$\eta_1 \equiv \mathsf{Computes}(X, G(\hat{A}, g^a, g^b)) \supset \exists A'.X = A'$$
$$\eta_2 \equiv \mathsf{Computes}(X, F'(\hat{Z}, g^{ab})) \supset \mathsf{Has}(X, g^{ab})$$
$$\eta_3 \equiv \diamondminus\mathsf{Fresh}(Z,x) \supset \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, g^x\}, G(\hat{A}, g^a, g^b))$$
$$\eta_4 \equiv \diamondminus\mathsf{Fresh}(Z,x) \wedge \diamondminus\mathsf{Fresh}(W,y) \supset \neg\mathsf{Contains}(\{\hat{X}, \hat{Y}, g^y, F(\hat{X}, g^x, g^y), F'(\hat{X}, g^{xy})\}, G(\hat{A}, g^a, g^b))$$
$$\eta_5 \equiv \diamondminus\mathsf{Fresh}(Z,x) \wedge \diamondminus\mathsf{Fresh}(W,y) \wedge \mathsf{Contains}(\{\hat{X}, \hat{Y}, G(\hat{X}, g^x, g^y), G'(\hat{X}, g^{xy})\}, G(\hat{A}, g^a, g^b)) \supset (\hat{X} = \hat{A} \wedge \wedge g^b = g^x \wedge g^a = g^y)$$

However, as mentioned before, this class of protocols does not have the matching conversations based authentication property for the initiator.

Design Tradeoffs Template AKE1 provides a stronger form of authentication: matching conversations for both initiator and responder whereas AKE2 only provides matching conversations for responder. Therefore, in AKE1, the initiator is required to reveal his identity in the first message of the protocol, and hence instances of this template cannot provide identity protection against active attackers for the initiator. Additionally, assumptions ϵ_1 and ϵ_3 together imply that when the function variable F is instantiated to the signature function, A has a non-repudiable proof of communication with B. In the logic, such a proof just

$A \to B : g^a$	$A \to B : g^a$	$A \to B : g^a$
$B \to A : g^b, SIG_B(g^b, g^a, A)$	$B \to A : g^b, E_{g^{ab}}(SIG_B(g^a, g^b))$	$B \to A : g^b, SIG_B(g^b, g^a), H_{g^{ab}}(B)$
$A \to B : SIG_A(g^a, g^b, B)$	$A \to B : E_{g^{ab}}(SIG_A(g^b, g^a))$	$A \to B : SIG_A(g^b, g^a), H_{g^{ab}}(A)$
ISO 9798-3 Key exchange	STS	Basic SIGMA

Figure 5. Instantiations of authenticated key-exchange templates

uses terms that A possesses (given by the Has predicate) and the honesty of B.

One of the design goals for instances of the AKE2 template such as SIGMA [19] was to provide identity protection for the initiator. Since the initiator A can only reveal his identity in the third message of the protocol, and cannot be sure that the responder B knows he is talking to A until B receives the last message, template AKE2 does not provide the strong authentication property for the initiator. A counterexample run is shown below:

$$\begin{array}{l} A \to I(B) : g^a \\ I(C) \to B : g^a \\ B \to I(C) : g^b, F(B, g^b, g^a), F'(B, g^{ab}) \\ I(B) \to A : g^b, F(B, g^b, g^a), F'(B, g^{ab}) \\ A \to I(B) : G(A, g^a, g^b), G'(A, g^{ab}) \end{array}$$

In this scenario, A believes he has completed a session with B, while B is waiting for the third message, thinking that he is engaged in a session with C. This counterexample also shows that A's transcript of the run cannot be used to prove that B was involved in the protocol. Hence this template does not provide non-repudiation even when F is instantiated to the signature function.

5. The STS-MQV Protocol Family

While previous sections have focused on formal reconstruction of known protocols, here we examine the possibility of synthesizing new protocols by reusing constructs from existing protocols. We begin with two well-known protocol families: the first includes Diffie-Hellman key exchange and several variants that enhance the key derivation function (MTI/A, UM and MQV); the second is the IKE family - STS being the base protocol and a few steps further on is JFKr. These two protocol families are combined to map out a two-dimensional matrix of protocols (see Figure 6) which, to the best of our knowledge, has not been previously studied in the open literature. An intuitive presentation of this derivation is in Appendix B. The focus here is on a core part of the synthesis that follows naturally from the abstraction and refinement methodology. Specifically, we formally prove authentication, shared secrecy and identity protection properties of the protocols in the first three columns of Figure 6. Formalizing the remaining security properties and derivation steps discussed in Appendix B is an interesting challenge, which we hope to address in future work.

5.1. Formal Synthesis

We identify two base protocol templates: a key computation template (KC) and an authenticated key exchange template (AKC). AKC relies on a key computation template with exactly the properties characterized by KC. The first column in Figure 6 contains protocols DH, MTI/A, UM and MQV which are instantiations of KC; the second column has instantiations of AKC, where each instantiated protocol uses the KC instantiation on its left (e.g., STS^{MQV} uses MQV). In Section 5.2, we prove the security properties of these protocols using the abstraction and instantiation methodology. The protocols in the third column are obtained from those in the second by applying an identity protection refinement. Formally, this involves combining the AKC template with an encryption template in a manner similar to Example 4.2.1 in Section 4.2. The proofs are deferred to the full version of the paper.

5.2. Characterizing and Combining KC and AKC

In this section, we discuss the main ideas used in characterizing and combining the KC and AKC templates.

Key Computation Template, KC. The template is characterized by a formula capturing the idea that a shared key has associated with it two public-private key pairs and in order to compute the key, it is necessary and sufficient to possess one private key and the other public key. Formally,

$$\begin{array}{l} \mathsf{Computes}(X, H(t_1, f(t_1), t_2, f(t_2))) \equiv \\ \quad \mathsf{Has}(X, (t_1, f(t_2))) \vee \mathsf{Has}(X, (t_2, f(t_1))) \end{array}$$

Here, H is a function variable denoting the key computation function while the variable f denotes the function that is used to compute the public key given the

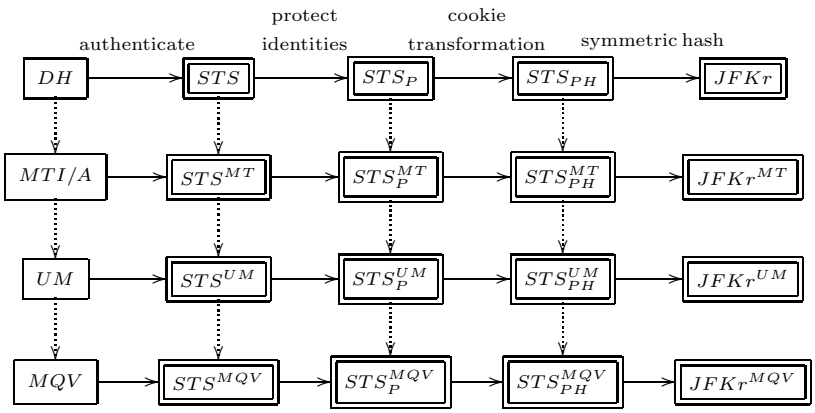

Figure 6. Design by Combining Templates

private key. It is easy to see that the key computation functions of all four protocols (DH through MQV) satisfy this hypothesis. For example, the Diffie-Hellman shared key g^{ab} is associated with the public-private key pairs (g^a, a) and (g^b, b) and computing it requires either (a, g^b) or (b, g^a). Note that the hypothesis for this template only characterizes the properties of certain functions. The proof that a particular instantiation has this property must therefore follow from axioms of the proof system and is independent of the protocol in which this template is used. This observation will be crucial when we subsequently combine this template with AKC.

Authenticated Key Exchange Template, AKC. AKC (given below) is a generalization of AKE2, the template which yielded STS and SIGMA in Section 4.3. AKE2 can be obtained by instantiating the function variables f and H to the Diffie-Hellman functions, F_1, F_1' to F, F', and G_1, G_1' to G, G'.

$$
\begin{array}{l}
A \rightarrow B : f(t_1) \\
B \rightarrow A : f(t_2), F_1(B, f(t_1), f(t_2)), \\
\qquad\quad F_1'(B, H(t_1, f(t_1), t_2, f(t_2))) \\
A \rightarrow B : G_1(A, f(t_2), f(t_1)), G_1'(A, H(t_1, f(t_1), t_2, f(t_2)))
\end{array}
$$

The invariants characterizating this template are similar to those for AKE2. The further abstraction arises from understanding two points. First, the essential property provided by the Diffie-Hellman terms is characterized by the KC template. Second, the role of function F in AKE2 is to guarantee that only B possesses one of the private keys required to compute the shared secret, while F' provides a proof that B actually possesses the key. These properties, captured by invariants about F_1 and F_1', yield a similar proof of the shared secrecy and authentication properties of AKC.

Combining the Templates Diffie-Hellman and MQV functions are instantiations of the KC template and, as noted earlier, since this template depends only on the functions, the property is guaranteed irrespective of the message structure of the specific protocol in which they are used. Instantiating the underlying KC template of AKC with Diffie-Hellman and using encrypted signatures over public keys gives us the STS protocol. As proved in Section 4.3, this protocol satisfies the assumed invariants of AKE2 and hence AKC. It therefore provides an authenticated shared secret.

$$
\begin{array}{l}
A \rightarrow B : g^x \\
B \rightarrow A : g^y, C_B, E_K\left(SIG_B(g^y, g^x)\right) \\
A \rightarrow B : C_A, E_K\left(SIG_A(g^x, g^y)\right)
\end{array}
$$

On the other hand, instantiating the KC template of AKC to MQV and using a combination of certified Diffie-Hellman keys and encryptions over the public keys, we get the new STS^{MQV} protocol. This protocol also satisfies the assumed invariants of the AKC template and therefore guarantees an authenticated shared secret. The substitutions for F_1 and F_1' are respectively the certificate G_B and the term encrypted with the MQV shared key.

$$
\begin{array}{l}
A \rightarrow B : g^x, g^a \\
B \rightarrow A : g^y, g^b, G_B, E_K\left(g^y, g^x\right) \\
A \rightarrow B : G_A, E_K\left(g^x, g^y\right)
\end{array}
$$

An advantage of this protocol over STS is that it is computationally more efficient, a property that it inherits from the MQV protocol (see Appendix B for further discussion).

6. Conclusions and Future Work

While there is ample evidence that protocol designers think systematically about protocol requirements and the means to achieve them (e.g., [1, 19]), it is a significant challenge to make these natural intuitions precise enough to provide systematic proofs of protocol properties. We believe that protocol templates, and an accompanying higher-order extension of our previous protocol logic, furnish some useful techniques for modular reasoning. In particular, similar protocols can now be proved to have identical or related properties using a single proof about a protocol template. Moreover, it is possible for multiple properties of a single protocol to be established using different templates for each property. While we have illustrated these general protocol proof methods with a few simple examples, we believe that many more templates and associated proofs can be devised.

The logical foundation for the proof method shown in this paper is the very simple idea of extending a protocol notation (cords) and protocol logic with function variables. This allows protocol templates to be written in a natural way, and allows them to be proved correct using implicit universal quantification over function variables.

We have developed some challenge-response and key exchange protocol templates that, in addition, required adding symmetric encryption and cryptographic hash to our previous protocol logic. We also used protocol templates to explore a design space surrounding JFK (Just Fast Keying) and related protocols from the IKE (Internet Key Exchange) family. This exploration revealed some trade-offs between authentication, identity protection, and non-repudiation; and produced some interesting protocols that appear not to have been previously studied in the open literature.

In future work, we hope to develop useful tool support for protocol derivation steps and the associated logic. A current effort underway draws on program derivation and verification experience at Kestrel Institute. The software infrastructure supporting protocol derivations is based on *especs* [31, 32, 4], a framework for refinement and automated composition of state machines, where states are annotated by algebraic specifications, and transitions by morphisms between them.

References

[1] W. Aiello, S. M. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 48–58. ACM Press, 2002.

[2] W. Aiello, S.M. Bellovin, M. Blaze, R. Canetti, A.D. Keromytis J. Ioannidis, and O. Reingold. Just fast keying (JFK), 2002. Internet draft.

[3] R. Ankney, D. Johnson, and M. Matyas. The unified model, 1995. Contribution to X9F1.

[4] M. Anlauff and D. Pavlovic. On specification carrying software, its refinement and composition. In H. Ehrig, B.J. Kramer, and A. Ertas, editors, *Proceedings of IDPT 2002*. Society for Design and Process Science, 2002.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on the Foundations of Computer Science*. IEEE, 2001. Full version available at `http://eprint.iacr.org/2000/067/`.

[6] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[7] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. Proceedings of Mathematical Foundations of Programming Semantics, to appear in *Electronic Notes in Theoretical Computer Science*, 2003.

[8] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (Extended abstract). In *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, pages 11–23, 2003.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[10] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[11] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[12] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proceedings of 14th IEEE Computer Security Foundations Workshop*, pages 241–255. IEEE, 2001.

[13] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):677–721, 2004.

[14] W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[15] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), 1998. RFC 2409.

[16] B. S. Kaliski, Jr. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):275–288, 2001.

[17] C. Kaufman. Internet Key Exchange (IKEv2) Protocol, 2004. Internet Draft.

[18] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology*, 7(2):79–130, 1994.

[19] H. Krawczyk. Sigma: The sign-and-mac approach to authenticated diffie-hellman and its use in the IKE protocols. In *Advances in Cryptology - CRYPTO 2003*, volume 2729, pages 400–425. Springer-Verlag Heidelberg, 2003.

[20] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. Technical Report 98-05, COOR, 1998.

[21] G. Lowe. Some new attacks upon security protocols. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE, 1996.

[22] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

[23] T. Matsumoto, Y. Takashima, and H. Imai. On seeking smart public-key distribution systems. *The Transactions of the IECE of Japan*, E69:99–106, 1986.

[24] C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of 7th IEEE Computer Security Foundations Workshop*, pages 84–89. IEEE, 1994.

[25] A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentication. In *Workshop on Selected Areas in Cryptography (SAC '95)*, pages 22–32, 1995.

[26] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[27] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report). In Stephen Brookes and Michael Mislove, editors, *17th Annual Conference on the Mathematical Foundations of Programming Semantics, Arhus, Denmark, May, 2001*, volume 45. Electronic notes in Theoretical Computer Science, 2001.

[28] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.

[29] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[30] L.C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[31] D. Pavlovic and D. R. Smith. Composition and refinement of behavioral specifications. In *Automated Software Engineering 2001. The Sixteenth International Conference on Automated Software Engineering.* IEEE, 2001.

[32] D. Pavlovic and D. R. Smith. Guarded transitions in evolving specifications. In H. Kirchner and C. Ringeissen, editors, *Proceedings of AMAST 2002*, volume 2422 of *LNCS*, pages 411–425. Springer Verlag, 2002.

[33] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, Washington, 2001.

[34] A. Ramanathan, J. C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *FOSSACS 2004 - Foundations of Software Science and Computation Structures*, March 2004.

A. Protocol Logic Extensions

Most of the predicates and axioms of protocol logic used in the formal proofs in this paper are presented in [6, 8]. The main extensions include the Computes predicate and some axioms for reasoning about symmetric encryption and cryptographic hash. The definition of Computes in terms of Has and the axioms it satisfies is presented in Table 3. Intuitively, **CP2** says that there are two ways an agent can possess some term: she can construct it from its components or she can receive it as a part of some message. Axiom **CP3** says that every term that appears on the network has a source: it originated from some process that actually computed the term. One use of Computes is to reason about the source of a term. This is useful for reasoning about authentication properties of protocols. A second use is to capture hardness assumptions about cryptographic primitives, which is important for reasoning about secrecy. For example, we postulate that the only way to compute a Diffie-Hellman secret is to have one exponent and the other exponential. Similarly, in order to compute an encrypted message, it is essential to possess the key and the plaintext. The axioms about symmetric encryption and cryptographic hash are deferred to the full version of the paper since the proofs in which they are used have also been omitted due to space constraints.

B. Deriving the STS-MQV Family

In this section, we present a derivation of the matrix of protocols summarized in Figure 6 in Section 5. This involves the derivation of $JFKr$ (the first row of the figure), the derivation of the Diffie-Hellman refinements (the first column of the figure which we discuss in Appendix B.1), and the combination of these two sets of protocols, discussed below in Appendix B.2. The interested reader is referred to [6] for a similar derivation of $JFKr$, starting from Diffie-Hellman and challenge-response components.

B.1. Refinements of Diffie-Hellman key exchange

In this section, we examine the structure of a family of protocols that has gone through the standardization

$$
\begin{array}{rl}
\textbf{CP1} & \mathsf{Computes}(X,t) \supset \mathsf{Has}(X,t) \\
\textbf{CP2} & \mathsf{Has}(X,t) \supset (\mathsf{Computes}(X,t) \vee \exists m.(\diamondsuit\, \mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m,t))) \\
\textbf{CP3} & (\diamondsuit\, \mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m,t)) \supset \\
 & \exists Y.\exists m'.(\mathsf{Computes}(Y,t) \wedge \diamondsuit\, \mathsf{Send}(Y,m') \wedge \mathsf{Contains}(m',t)) \\
\mathsf{Computes}(X,t) \equiv & (t = g^{ab} \wedge \mathsf{Computes}_{\mathsf{DH}}(X, g^a b)) \vee \\
 & (t = H(a) \wedge \mathsf{Computes}_{\mathsf{HASH}}(X, H(a))) \vee \\
 & (t = E_a(b) \wedge \mathsf{Computes}_{\mathsf{ENC}}(X, E_a(b))) \\
\mathsf{Computes}_{\mathsf{DH}}(X, g^{ab}) \equiv & ((\mathsf{Has}(X,a) \wedge \mathsf{Has}(X,g^b)) \vee (\mathsf{Has}(X,b) \wedge \mathsf{Has}(X,g^a))) \\
\mathsf{Computes}_{\mathsf{ENC}}(X, E_a(b)) \equiv & \mathsf{Has}(X,a) \wedge \mathsf{Has}(X,b) \\
\mathsf{Computes}_{\mathsf{HASH}}(X, H(a)) \equiv & \mathsf{Has}(X,a)
\end{array}
$$

Table 3. Computes **Axioms**

process. Starting from the standard Diffie-Hellman protocol, we successively obtain protocols with more desirable security properties as the derivation proceeds. The derivation steps are shown in Figure 7. The properties of interest include key secrecy, implicit authentication, forward secrecy, known-key security, resistance to unknown key share attacks, and efficiency.

Ephemeral Diffie-Hellman DH **key exchange** The basic Diffie-Hellman protocol [9] provides a way for two parties to set up a shared key (g^{xy}) which a passive attacker cannot recover.

$$A \rightarrow B : g^x$$
$$B \rightarrow A : g^y$$

$$k = g^{xy} = (g^y)^x = (g^x)^y$$

There is no authentication guarantee: the secret is shared between two parties, but neither can be sure of the identity of the other. One way to overcome this is for participants to have their public Diffie-Hellman values certified by a trusted authority (static Diffie-Hellman) and use those keys instead in the exchange. But, in that case, A and B would compute the same shared secret in every session, i.e, the protocol would not have known-key security.

MTI/A **key exchange** The MTI/A protocol [23] tries to achieve authenticated key exchange by combining ephemeral and static Diffie-Hellman.

$$A \rightarrow B : g^x, g^a, G_A$$
$$B \rightarrow A : g^y, g^b, G_B$$

$$k = g^{ay+bx} = (g^y)^a (g^b)^x = (g^x)^b (g^a)^y$$

It is assumed that parties A and B have long term Diffie-Hellman exponents a and b and have obtained certificates G_A and G_B for corresponding public exponentials g^a and g^b. Also, x and y are generated fresh for every session and so a unique shared key is generated each time. It therefore provides known-key security, key secrecy, and implicit authentication. However, there is no forward secrecy since if the long-term secrets a and b are revealed, an attacker can compute all past session keys. Also, this protocol is open to an unknown key-share attack if an attacker can obtain certificates for exponentials of his choice without having to prove that he possesses the corresponding private exponents. The attack was first presented in [25].

UM **key exchange** The *Unified model* protocol [3] represents another step forward. It combines ephemeral and static Diffie-Hellman in a very simple manner: the shared secret is just a concatenation of the ephemeral and static shared secrets.

$$A \rightarrow B : g^x, g^a, G_A$$
$$B \rightarrow A : g^y, g^b, G_B$$

$$k = g^{ab} || g^{xy}$$

Besides providing the security guarantees given by MTI/A, it also provides perfect forward secrecy since the ephemeral shared secrets cannot be computed even if the long term private keys are revealed. However, this protocol is also open to an unknown key-share attack.

MQV **key exchange** The final protocol in the derivation is MQV [20].

$$A \rightarrow B : g^x, g^a, G_A$$
$$B \rightarrow A : g^y, g^b, G_B$$

$$k = g^{(ag^x+x)(bg^y+y)}$$

It provides all but one of the desirable properties: key secrecy, implicit authentication, known-key security, forward secrecy, and computational efficiency. Note however that it is open to an unknown key-share attack as pointed out in [16].

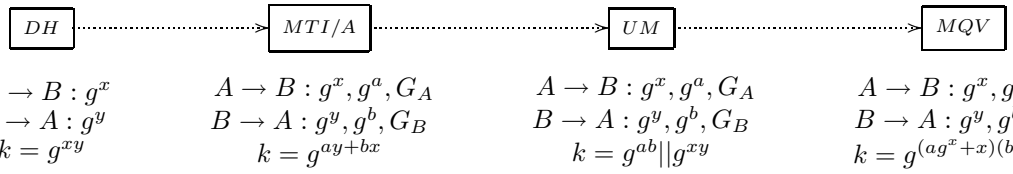

Figure 7. Refinements of the Diffie-Hellman key exchange

B.2. Combining the Derivations

In this section, we examine how security protocols can be constructed by combining derivations. Specifically, the Diffie-Hellman component in the $JFKr$ derivation is replaced by a more "refined" component from the second derivation. This yields a class of protocols which inherit security properties from both the derivations. The derivation graph for the complete class is shown in Figure 6. Due to space constraints, we examine only one path in detail to get a sense of the general method.

B.2.1. Derivation of the JFK_r^{MQV} protocol We start by composing the MQV component with the challenge-response protocol. Since the MQV shared secret includes certified static Diffie-Hellman exponentials, signatures are no longer necessary, resulting in protocols with less computational overhead.

Protocol STS^{MQV} By composing the MQV key exchange with the challenge response protocol, we obtain the STS^{MQV} protocol. Fresh values m and n are instantiated to pairs of Diffie-Hellman exponents (g^x, g^a) and (g^y, g^b), and the key K is instantiated to a key generated from the MQV shared secret.

$$\begin{array}{l} A \to B : g^x, g^a \\ B \to A : g^y, g^b, G_B, E_K\left(g^y, g^x\right) \\ A \to B : G_A, E_K\left(g^x, g^y\right) \end{array}$$

This protocol inherits the key secrecy and mutual authentication properties from STS and forward secrecy, known-key security, and computational efficiency from MQV. Note that the unknown key-share attack on MQV goes away because STS provides explicit key authentication (not just implicit).

Protocol STS_P^{MQV} In order to protect the identity of the participants against a passive attacker, we move the certificates inside the encryption. The resulting protocol is denoted by STS_P^{MQV}.

$$\begin{array}{l} A \to B : g^x, g^a \\ B \to A : g^y, g^b, E_K\left(G_B, g^y, g^x\right) \\ A \to B : E_K\left(G_A, g^x, g^y\right) \end{array}$$

While preserving the security properties achieved in the previous step, this protocol, in addition, provides a form of identity protection. The identities of both the participants are protected against passive attackers, while the identity of the initiator A is also protected against active attackers. Notice that the identity protection this protocol provides is much weaker than that of STS_P. Public keys are sent in the clear, and an attacker can deduce the identity of a participant if he possesses his certificate. This drawback is the result of using encryptions only, and not encrypted signatures. We cannot move public keys into the encryption since A needs to know g^b in order to generate the shared key K.

Protocol STS_{PH}^{MQV} In order to make the protocol resistant to blind denial of service (DoS) attacks, we preform the *cookie transformation* [6]. The resulting protocol is denoted by STS_{PH}.

$$\begin{array}{l} A \to B : g^x, g^a \\ B \to A : g^y, g^b, HMAC_{BH}(g^y, g^x) \\ A \to B : g^x, g^a, g^y, g^b, HMAC_{BH}(g^y, g^x), E_K\left(G_A, g^x, g^y\right) \\ B \to A : E_K\left(G_B, g^y, g^x\right) \end{array}$$

The other security properties are preserved under this transformation.

Protocol $JFKr^{MQV}$ Finally, we add message authentication codes to obtain $JFKr^{MQV}$ protocol.

$$\begin{array}{l} A \to B : g^x, g^a \\ B \to A : g^y, g^b, HMAC_{BH}(g^y, g^x) \\ A \to B : g^x, g^a, g^y, g^b, HMAC_{BH}(g^y, g^x), E_K\left(G_A, g^x, g^y\right) \\ \qquad HMAC_{K'}\left(A, E_K\left(G_A, g^x, g^y\right)\right), \\ B \to A : E_K\left(G_B, g^y, g^x\right), \\ \qquad HMAC_{K'}\left(B, E_K\left(G_B, g^y, g^x\right)\right) \end{array}$$

The final protocol provides key secrecy, mutual authentication, forward secrecy, known-key security, computational efficiency, identity protection and DoS protection, inheriting most of the positive properties from the parent derivations.