Deriving Secrecy in Key Establishment Protocols

Dusko Pavlovic[1] and Catherine Meadows[2]

[1] Kestrel Institute, Palo Alto, CA 94304
dusko@kestrel.edu
[2] Naval Research Laboratory, Washington, DC 20375
meadows@itd.nrl.navy.mil

Abstract. Secrecy and authenticity properties of protocols are mutually dependent: every authentication is based on some secrets, and every secret must be authenticated. This interdependency is a significant source of complexity in reasoning about security. We describe a method to simplify it, by encapsulating the authenticity assumptions needed in the proofs of secrecy. This complements the method for encapsulating the secrecy assumptions in proofs of authenticity, presented in [1]. While logically straightforward, this idea of encapsulation in general, and the present treatment of secrecy in particular, allow formulating scalable and reusable reasoning patterns about the families of protocols of practical interest. The approach evolved as a design strategy in the Protocol Derivation Assistant (Pda), a semantically based environment and toolkit for derivational approach to security [2,3].

1 Introduction

All secure communication on public networks begins with key establishment. Many diverse key distribution and key agreement schemes have evolved, with subtle, complex, and sometimes unclear security properties Since they are critical for the functioning of the networks, it is desirable to establish their provable, rather than just empirical security.

1.1 Derivational Approach to Security

Practical methods for proving security evolve slowly. While most branches of engineering largely consist of methodologies for building complex systems from simple components, formulating the incremental and compositional methods for security engineering has been a challenging task: in general, security properties are not preserved under composition.

Although not straightforward, the progress towards methods for the incremental design and analysis of security protocols has been steady. The present work is a part of a continued effort towards capturing and formalizing a sound part of the incremental practices of security engineering, and supporting it in a semantically based integrated development environment for secure systems [2,3]. The

D. Gollmann, J. Meier, and A. Sabelfeld (Eds.): ESORICS 2006, LNCS 4189, pp. 384–403, 2006.

logic of [4,5] is a protocol analysis logic which incorporates incremental practices of security engineering such as refinement and composition. In [6] we developed a streamlined and simplified version of the authentication fragment of the logic in [4,5], that was still sufficient to uncover a flaw an IETF standardized protocol, that had already undergone extensive formal analysis. This led us to the idea, presented in [1], that many authentication proofs can be simplified, by encapsulating the needed secrecy requirements, and leaving them as open assumptions, to be discharged in separate proof modules. In the present work we present a secrecy logic, intended to support these separate proofs of secrecy properties. This time, the authentication assumptions, needed for secrets, are encapsulated, and left as open assumptions.

1.2 Encapsulation Trick

An important source of complexity in reasoning about security is the mutual dependancy of secrecy and authenticity: every secret must be authenticated, and every authentication must be based on a secret. This feedback loop of positive and negative knowledge statements, logical weaving and interleaving of authentication and secrecy, often generates confusion. The method of *encapsulation* allows us to untie some such knots, and to make some general patterns of formal reasoning actually and unexpectedly *simpler* then the informal counterparts, which were their source.

In previous work [1], we simplified authenticity proofs by encapsulating the secrecy assumptions on which they depended: e.g., an assumption that a key, used for authentication, was uncompromised, was simply left open, to be discharged in a separate proof module. A more common approach would have been to unfold the proof that the key remains secret within the given protocol. This proof would require an assumption that the key was authenticated — and this assumption would then usually be left open, as an assumption about the infrastructure. The authentication performed in the analyzed protocol would thus be reduced to another authentication, performed in the preceding key establishment protocol. While this approach is reasonable, unfolding a secrecy proof within each authenticity proof, and vice versa, does seem to complicate things, and does not seem necessary. In some cases, reducing authentications to the relevant secrecy assumptions, that encapsulate their own authenticity assumptions, often allows considerably simpler, and more insightful proofs.

Outline of the Paper. We present a method to derive secrecy properties of key establishment protocols, while assuming, and encapsulating for later implementation, the needed authentication properties of their components. Section 2 opens the exposition with an informal overview of the counterpart of this approach, where the authentication properties are derived, while the secrecy properties are assumed and encapsulated. An outline of the general framework used in protocol derivations is in the Appendix. Section 3 introduces the relations needed for modeling secrecy, and the derivation rules needed for proving it. Section 4 provides a method for proving a family of inductive statements to which the crucial

rule reduces the secrecy statements. Section 5 presents the basic derivation templates for two key establishment patterns: key distribution and key agreement. Section 6 concludes the paper with a summary of its contributions.

2 Overview of Authentication with Encapsulated Secrecy

The goal of an authentication protocol $\mathcal{Q}$ for a group G is to realize the authentication predicate

$$\mathsf{Auth}^{\mathcal{Q}}(G) = \forall XY \in G.\ \overline{L}^{\mathcal{Q}}_X \approx \overline{L}^{\mathcal{Q}}_Y$$

where $\overline{L}^{\mathcal{Q}}_X$ is the *complete view* of the principal X at the final state of the protocol $\mathcal{Q}$. The relation $\approx$ requires the views to be equal, except for the last send-receive pair, which the sender cannot ascertain. The predicate $\mathsf{Auth}^{\mathcal{Q}}(G)$ thus formalizes entity authentication through "matching histories", introduced in [7] and formalized in [8]. Message authentication can be captured in a similar way, by requiring that principals' records match when restricted to the actions containing the relevant term:

$$\mathsf{Auth}^{\mathcal{Q}}(t; G) = \forall XY \in G.\ \left(\overline{L}^{\mathcal{Q}}_X \restriction t\right) \approx \left(\overline{L}^{\mathcal{Q}}_Y \restriction t\right)$$

To represent, say, a freshly generated value, we take t to be a variable x bound by an action (νx) in $\mathcal{Q}$. Its propagation is tracked through the send, receive and assign actions in the various runs, and its authenticity means that the various principals' views of this coincide at the final state of each run. For a set of terms Θ, we abbreviate $\mathsf{Auth}(\Theta; G) = \bigwedge_{t\in\Theta} \mathsf{Auth}(t; G)$.

In general, the complete view $\overline{L}^{\sigma}_X$ of the principal X at the state σ is obtained by applying the available *authentication axioms* and logical rules to the (incomplete) view L^{σ}_X, which consists of the actions observed by X up to σ. The views grow as the runs progress. Each principal directly observes only her own actions (and the actions of her subprincipals); but the authentication axioms allow her to draw conclusions about the actions of others, roughly in the form: "If I am receiving this message, someone must have sent it."

The authentication axioms are subsumed under the authentication template

$$A:\ \forall x.\ \Phi(x) \wedge ((f^{AB}x))_A \Longrightarrow \Psi(x) \wedge \langle\langle f^{AB}x\rangle\rangle_{B<} < ((f^{AB}x))_A \qquad (\mathsf{au})$$

instantiated to particular formulas Φ and Ψ. Here $((t))_A$ means *"A receives a message containing t"*, and $\langle\langle t\rangle\rangle_{A(<)}$ means *"A sends (originates) a message containing t"* [6]. The prefix "A :" means that the schema is used in A's local reasoning. The most important instance of this schema is the challenge-response axiom

$$A:\ (\nu x)_A\Big(\langle\langle c^{AB}x\rangle\rangle_A < ((r^{AB}x))_A \\ \Longrightarrow \langle\langle c^{AB}x\rangle\rangle_A < ((c^{AB}x))_B < \langle\langle r^{AB}x\rangle\rangle_{B<} < ((r^{AB}x))_A\Big) \qquad (\mathsf{cr})$$

obtained by setting

$$\begin{aligned} f^{AB}x &= r^{AB}x \\ \Phi(x) &= \mathsf{New}(x) \wedge \langle\langle c^{AB}x\rangle\rangle_A < ((r^{AB}x))_A \\ \Psi(x) &= ((c^{AB}x))_B < \langle\langle r^{AB}x\rangle\rangle_B \end{aligned}$$

in (au), and abbreviating "$\forall x.\ \mathsf{New}(x) \Rightarrow$" to "$(\nu x)$". This axiom allows a principal A, who can only observe her own actions, to draw conclusions about B's actions, using the assumption that no one but B could have transformed $c^{AB}x$ to $r^{AB}x$. Axiom (cr) should thus be construed as a specification of the property of the challenge-response functions c and r, required for the authentication. This property is simply asserted (postulated) for the abstract challenge-response protocol, but the task is to refine this protocol, and implement c and r as more concrete cryptographic functions, which come with their own axioms, from which (cr) can then be derived as a theorem [6,1]. This is where the encapsulated *secrecy assumptions* enter scene. For instance, when the challenge-response functions are implemented as

$$\begin{aligned} c^{AB}x &= x \\ r^{AB}x &= S^B(A, x) \end{aligned}$$

where S^B is B's signature, then axiom (cr) is implied by the statement that this signature is B's secret, i.e. that only he can generate it:

$$A : \langle\langle S^B x\rangle\rangle_{X<} \Longrightarrow X = B$$

And this, furthermore, can only be true if the session where B's signature is established has been authenticated, so that no other principal can come in possession of his signature. And that authentication depended on some previous secrets, and so on.

In general, given a protocol refinement $\mathcal{Q}(f) \longrightarrow \mathcal{Q}(F/f)$, where an abstract function f is implemented as a more concrete function F, refining the authentication proof often requires an additional assumption that the function F, used to authenticate group G, is G's secret

$$\mathsf{Auth}^{\mathcal{Q}(f)}(G) \wedge \mathsf{Secr}^{\mathcal{Q}(F/f)}(F; G) \Longrightarrow \mathsf{Auth}^{\mathcal{Q}(F/f)}(G)$$

For instance, when the abstract response function r^{AB} is refined to signature S^B, axiom (cr) will be satisfied because S^B is B's secret, and no one else could generate his signed response.

Similarly, in order to derive a secrecy property of a refinement F of an abstract operation f in a protocol $\mathcal{Q}$, we shall often need the authenticity assumption, in the form

$$\mathsf{Secr}^{\mathcal{Q}(f)}(f; G) \wedge \mathsf{Auth}^{\mathcal{Q}(F/f)}(F; G) \Longrightarrow \mathsf{Secr}^{\mathcal{Q}(F/f)}(F; G)$$

The goal of the rest of the paper is to make this precise.

3 Modeling Secrecy

In this section we introduce the conceptual and notational infrastructure needed for the formal definition of secrecy and the rules for deriving it. There are several layers of structure, and the tempo is brisk: the details must be left for the subsequent sections.

3.1 Order and Security

The simplest process model suitable for capturing the various aspects of security seems to be the based on partial orders, or more precisely partially ordered multisets (pomsets) [9]. It extends the trace based models, such as strand spaces [10], and simplifies the process-calculus based models.

While authenticity is achieved through partial ordering of actions, secrecy is concerned with the *computability* relation between terms, as they propagate through communication. The abstract pomset model, used for deriving authenticity, is now extended by an abstract computability relation. This will suffice for secrecy derivations. The model (outlined in the Appendix) consisted of:

- partial order of *terms* and subterms $(\mathcal{T}, \sqsubseteq)$,
- partial order of *principals* and subprincipals $(\mathcal{W}, \Subset)$,
- set of *actions* $\mathcal{A}$ generated over the terms,
- *processes* as partially ordered multisets of actions [9], i.e. maps $\mathbb{L} \xrightarrow{L} \mathcal{A} \times \mathcal{W}$, where $\mathbb{L}$ is a partial order, and
- *runs*, which extend processes by assigning to each receive action a unique send action

Now we add:

- partial order $\Gamma \vdash \Theta$ between finite sets $\Gamma, \Theta \subseteq \mathcal{T}$, meaning that each term $t \in \Theta$ can be computed from a tuple[1] of terms $\boldsymbol{g} \in \Gamma$.

In the present paper, we study the useful symbolic interpretations of computability relation.

3.2 Symbolic Computability

In general, the computability relation among the terms used in a protocol can be given by rewrite rules, e.g.

$$x, y \vdash \langle x, y \rangle \qquad \langle x, y \rangle \vdash x, y \qquad k, x \vdash Ekx \qquad k, Ekx \vdash x$$

where $\langle x, y \rangle$ represents a pairing operation, and Ekx the encryption of x by k. Abadi and Rogaway [11] use such computability relation. Paulson's **analz** operator [12] corresponds to the closure $\Gamma^{\vdash} = \{t \in \mathcal{T} \mid \Gamma \vdash t\}$ for the special case of the theory of encryption and tupling. More generally, given an algebraic theory

[1] We often abbreviate $g_1, g_2, \ldots, g_n$ to $\boldsymbol{g}$.

T with a signature Σ_T, and the set of derived operations $\overline{\Sigma}_T$, the computability relation can be defined by

$$\Gamma \vdash \Theta \iff \forall t \in \Theta \exists \boldsymbol{g} \in \Gamma \exists \varphi \in \overline{\Sigma}_T.\ T \models (t = \varphi \boldsymbol{g})$$

In words, for every element $t \in \Theta$ we can find a tuple $\boldsymbol{g} \in \Gamma$ and an algebraic operation φ, such that the equation $t = \varphi \boldsymbol{g}$ can be proven in algebra T. The rewrite rules given above are obtained for the algebraic theory T over the signature $\Sigma_T = \{\langle -, - \rangle, \pi_1, \pi_2, E, D\}$ and the equations

$$\pi_1\langle x, y\rangle = x \qquad \pi_2\langle x, y\rangle = y \qquad Dk(Ekx) = x$$

where we make the restriction that π_i is applied only to the result of concatenation and Dk is applied only to the result of applying Ek. [2] Note that the encryption E and decryption D are presented as *curried* binary operations on keys and messages. This will simplify notation in the sequel: e.g. the encryption and decryption by a key k become unary operations

Notation. It is convenient to extend the computability relation from the elements of $\mathcal{T}$ to functions $f : \mathcal{T} \longrightarrow \mathcal{T}$, as follows:

$$\begin{aligned} \Gamma \vdash f &= \forall x.\ \Gamma, x \vdash fx \\ \Gamma \vdash f^{-1} &= \forall x.\ \Gamma, fx \vdash x \\ \Gamma, f \vdash t &= \forall \Xi.\ \Gamma, \Xi \vdash f \Rightarrow \Gamma, \Xi \vdash t \\ \Gamma, f^{-1} \vdash t &= \forall \Xi.\ \Gamma, \Xi \vdash f^{-1} \Rightarrow \Gamma, \Xi \vdash t \end{aligned}$$

This extends to sets of functions in the obvious way. E.g., if $E_G = \{E_X : \mathcal{T} \to \mathcal{T} | X \in G\}$ is a family of public key encryptions for the principals from some group $G \subseteq \mathcal{W}$, then $\Gamma \vdash E_G^{-1}$ means that the keys to decrypt the messages encrypted by any of $E_X \in E_G$ can be computed from Γ.

For an algebraic theory T with signature Σ_T, the above definition of computability implies that $x \vdash fx$, i.e. $\vdash f$, holds for every $f \in \Sigma_T$.

Finally, each order relation induces a closure operator on sets of terms:

$$\Gamma^{\vdash} = \{t \in \mathcal{T} \mid \Gamma \vdash t\} \qquad \Gamma^{\sqsupseteq} = \{t \in \mathcal{T} \mid \exists u \in \Gamma.\ u \sqsupseteq t\}$$

3.3 Guard Relation

A set of sets of terms $\mathcal{G} \in \wp\wp\mathcal{T}$ is said to *guard* a term t with respect to a set of terms $\Upsilon \subseteq \mathcal{T}$ if every computation of t from Υ must traverse some $\Gamma \in \mathcal{G}$, i.e.

$$\mathcal{G}\ \mathsf{guards}_{\Upsilon}\ \Theta \quad = \quad \forall t \in \Theta \exists \Gamma \in \mathcal{G} \forall \Xi \subseteq \Upsilon.\ \Xi \vdash t \Rightarrow \Xi \vdash \Gamma$$

We say that $\mathcal{G}$ guards Θ in a process L if it guards it with respect to the set of terms $\Upsilon = \mathcal{T}_L$ which become computable to all observers in any run of L.

[2] We note that without such restrictions, not only are the theories not equivalent, but as pointed out in [13], it is possible to have protocols that are secure in the rewrite theory that are not secure in the algebraic theory.

This notion is implicit in many symbolic analyses of secrecy. Since the environment Υ depends on the possible runs of L (of which there can be many!), proving that a term is guarded can be complex. Our approach is to prove that guardedness is satisfied if certain syntactic conditions on the runs that are easy to verify using the authentication logic are also satisfied. This allows us to encapsulate the complex secrecy proofs as well. A framework for a large class of such proofs is presented in section 4.

3.4 Secrecy Predicates and Rules

The information available to a principal $A \in \mathcal{W}$ at a state[3] σ in a run ℓ of a process $\mathbb{L} \xrightarrow{L} \mathcal{A} \times \mathcal{W}$ is conveniently subdivided into

- a *view* $L^\sigma_A : \mathbb{L}^\sigma_A \longrightarrow \mathcal{A} \times \mathcal{W}_{\Subset A}$, which is the restriction of the run $L = \langle L_{\mathcal{A}}, L_{\mathcal{W}} \rangle : \mathbb{L}^\ell \longrightarrow \mathcal{A} \times \mathcal{W}$ to the actions executed before the state σ by A's subprincipals, i.e. to the subposet $\mathbb{L}^\sigma_A = \{\xi \in \mathbb{L}^\ell \mid \xi \leq \blacktriangledown_\sigma \wedge L_{\mathcal{W}}\xi \Subset A\}$, and
- an *environment* Γ^σ_A, which consists of the fresh variables from σ^2_A and the terms which occur in σ^3_A, which together capture all terms that A has generated, received or computed up to the state σ.

Secrecy of a set of terms Θ for a group $G \subseteq \mathcal{W}$ at a state σ is then defined[4]

$$\begin{aligned}
\mathsf{Have}^\sigma(\Theta; G) &= \forall X \in G.\ \Gamma^\sigma_X \vdash \Theta \\
\mathsf{Only}^\sigma(\Theta; G) &= \forall X \in \mathcal{W} \forall t \in \Theta.\ \Gamma^\sigma_X \vdash t \Longrightarrow X \in G \\
\mathsf{Secr}^\sigma(\Theta; G) &= \mathsf{Have}^\sigma(\Theta; G) \wedge \mathsf{Only}^\sigma(\Theta; G) \\
&= \forall X \in \mathcal{W} \forall t \in \Theta.\ \Gamma^\sigma_X \vdash t \iff X \in G
\end{aligned}$$

Lemma 1. *(a) For* $\Phi \in \{\mathsf{Have}, \mathsf{Only}, \mathsf{Secr}\}$ *holds*

$$\Phi(\Theta_1; G) \wedge \Phi(\Theta_2; G) \iff \Phi(\Theta_1 \cup \Theta_2; G)$$

and therefore

$$\Theta_1 \supseteq \Theta_2 \Longrightarrow \Phi(\Theta_1; G) \Rightarrow \Phi(\Theta_2; G)$$

(b) Furthermore

$$\begin{aligned}
\mathsf{Have}(\Theta; G_1) \wedge \mathsf{Have}(\Theta; G_2) &\iff \mathsf{Have}(\Theta; G_1 \cup G_2) \\
\mathsf{Only}(\Theta; G_1) \wedge \mathsf{Only}(\Theta; G_2) &\iff \mathsf{Only}(\Theta; G_1 \cap G_2)
\end{aligned}$$

and therefore

$$G_1 \supseteq G_2 \Longrightarrow \mathsf{Have}(\Theta; G_1) \Rightarrow \mathsf{Have}(\Theta; G_2) \wedge \mathsf{Only}(\Theta; G_2) \Rightarrow \mathsf{Only}(\Theta; G_1)$$

[3] The definitions are in the Appendix.

[4] Note that $\mathsf{Only}(\Theta; G)$ is logically equivalent to $\forall X \in \mathcal{W}(\exists t \in \Theta.\ \Gamma^\sigma_X \vdash t) \Longrightarrow X \in G$.

Notation. It will be convenient to introduce the relation of *relative* computability

$$\Xi \vdash_A^\sigma \Theta = \Xi, \Gamma_A^\sigma \vdash \Theta$$

We write $\Xi \vdash_A^{\mathcal{Q}} \Theta$ and $\Xi \vdash_A \Theta$ for computability in all runs of a protocol $\mathcal{Q}$. We also elide the empty set, and write $\vdash_A^\sigma \Theta$ instead of $\emptyset \vdash_A^\sigma \Theta$. For a group $G \subseteq \mathcal{W}$, we write

$$\Xi \vdash_G \Theta = \forall X \in G.\ \Xi \vdash_X \Theta$$

Rules. The basic steps in the derivations of secrecy will be

$$\frac{\mathsf{Have}^\sigma(\Xi; G) \qquad \Xi \vdash_G^\sigma \Theta}{\mathsf{Have}^\sigma(\Theta; G)}\ (\mathsf{have}) \qquad \frac{\mathsf{Only}^\sigma(\Xi_i; G_i)\ \big|_{i=1}^n \qquad \{\Xi_i\}_{i=1}^n \text{ guards } \Theta}{\mathsf{Only}^\sigma(\Theta; \bigcup_{i=1}^n G_i)}\ (\mathsf{only})$$

$$\frac{\mathsf{Secr}^\sigma(\Xi_i; G_i)\ \big|_{i=1}^n \qquad \Xi_i \vdash_{G_i}^\sigma \Theta\ \big|_{i=1}^n \qquad \{\Xi_i\}_{i=1}^n \text{ guards } \Theta}{\mathsf{Secr}^\sigma(\Theta; \bigcup_{i=1}^n G_i)}\ (\mathsf{secr})$$

4 Proving Guards

In this section, we explore the ways in which the guarding assumptions of (only) and (secr) can often be proved.

In practice, guards are often realized using functions which are hard to invert, or functions which are easy to invert with a key, but hard to invert without it. In modern cryptography, such functions are developed as *one-way* functions, and as *trapdoor* functions respectively [14, 2.4]. Very roughly, the idea is that the output of a one-way function does not contribute to computations that may yield its input

$$\Gamma, Fk \vdash k \Longrightarrow \Gamma \vdash k$$

whereas a trapdoor function allows computing a part m of its input only if another part k, called *trapdoor*, is also computable

$$\Gamma, Ekm \vdash m \Longrightarrow \Gamma \vdash k \ \vee\ \Gamma \vdash m$$

However, these formulations abstract away the fact that a function may not be invertible on its own, but may become invertible in a context, via equations like $Dk(Ekx) = x$. The definitions that we propose below capture this fact, but remain an algebraic approximation, inevitably crude, of concepts that essentially involve probabilistic computation. However, this initial simplification seems necessary for incremental proofs of secrecy, and open for refinements to more precise models.

4.1 Guarding by One-Way Functions

Fix a term algebra $\mathcal{T}$, given with a subterm relation $\sqsubset$, a set of function symbols Σ, and a computability relation $\vdash$.

If $f \in \Sigma$ is a function symbol of arity n and $i \leq n$, we call a pair $\langle f, i\rangle$ a *position*, and denote by $f(\sigma)_i$ a term in the form $fs_1s_2 \dots s_{i-1}\sigma s_{i+1} \dots s_n$, i.e. where σ occurs as i-th argument of f.

Definition 1. *Given sets $\Upsilon, \Theta \subseteq \mathcal{T}$ of terms and a set $\Pi \subseteq \Sigma \times \mathbb{N}$ of positions, we extract the Υ-terms which occur in Θ just in the Π-positions by the operator*

$$\mathsf{par}_\Upsilon(\Theta, \Pi) = \big\{\sigma \in \Upsilon \setminus \Theta \;\mid\; \forall t \in \Theta \forall s \sqsubseteq t \forall f \in \Sigma \forall i \in \mathbb{N}.\; s = f(\sigma)_i \;\Rightarrow\; \langle f, i\rangle \in \Pi\big\}$$

The context Υ is often all of the term algebra $\mathcal{T}$, or the set $\mathcal{T}_L^\sqsupseteq$ of the subterms from the set $\mathcal{T}_L$ of the terms that may be sent in runs of the process L. We write $\mathsf{par}_L(\Theta, \Pi) = \mathsf{par}_{\mathcal{T}_L^\sqsupseteq}(\Theta, \Pi)$, and $\mathsf{par}(\Theta, \Pi) = \mathsf{par}_{\mathcal{T}}(\Theta, \Pi)$.

Definition 2. *A term σ is purged from a position $\langle f, i\rangle$ in the terms of Θ by replacing in every $t \in \Theta$, all occurrences of σ as i-th argument of f by a fresh variable x, not occurring in Θ. Formally,*

$$\mathsf{pur}_\sigma(\Theta, \langle f, i\rangle) = \big\{t[f(x)_i] \mid t[f(\sigma)_i] \in \Theta\big\}$$

where $t[f(\sigma)_i]$ displays all the occurrences of $f(\sigma)_i$ in t. Then $\mathsf{pur}_\Upsilon(\Theta, \Pi)$ is obtained by sequentially purging all $\sigma \in \Upsilon$ the $\sqsubset$-maximal ones first.

Note that, if σ never occurs in a position from Π, then $\mathsf{pur}_\Upsilon(\Theta, \Pi) = \emptyset$.

Lemma 2. *The operation $\mathsf{pur}_\Upsilon(\Theta, \Pi)$ is well defined: the order of purges of the individual terms $\sigma \in \Theta$ does not matter, as long as the maximal terms are purged first.*

Proof. This follows from the fact that two terms are either disjoint, or one is entirely contained in another.

Definition 3. *A position set Π is said to be one-way if it satisfies*

$$\mathsf{Onwy}(\Pi) = \forall \Theta\; \forall \sigma \in \mathsf{par}(\Theta, \Pi).\;\; \Theta \vdash \sigma \;\;\Longrightarrow\;\; \mathsf{pur}_\sigma(\Theta, \Pi) \vdash \sigma$$

We write $\mathsf{Onwy}(f, i)$ instead of $\mathsf{Onwy}(\{\langle f, i\rangle\})$, and we write $\mathsf{Onwy}(f)$ instead of $\mathsf{Onwy}(\{\langle f, 1\rangle, \dots, \langle f, n\rangle\})$, where f is an n-ary function symbol.

Lemma 3. *For the function E, with the computability relation as defined in sec. 3.2, holds $\mathsf{Onwy}(E, 1) \wedge \neg\mathsf{Onwy}(E, 2)$.*

Proof. The second conjunct is a consequence of the rewrite $k, Ekm \vdash m$. For the proof of the first, let $\mathcal{D} = \Gamma_1 \vdash \dots \vdash \Gamma_n \vdash \sigma \in \mathsf{par}(\Theta, \langle E, 1\rangle)$ where $\Gamma_1 \subseteq \Theta$ and each $\vdash$-step consists of a single application of a rewrite rule. We want to show that if $\mathcal{D}$ is a derivation, then so is the result of applying the purge function to each term in $\mathcal{D}$. The proof will be by induction on the length of $\mathcal{D}$. The result clearly holds when the length of $\mathcal{D}$ is one. Suppose the result holds for length less than n. Suppose $\mathsf{pur}_\sigma(\Theta, \Pi) \nvdash \sigma$. Then one of the steps Γ_i in $\mathcal{D}$ must be of the form $S \cup \{\sigma, E\sigma y\} \vdash S \cup \{\sigma, E\sigma, y\}$ or $S \cup \{\sigma, y\} \vdash S \cup \{\sigma, y, E\sigma y\}$. Then $\Gamma_1 \vdash \dots \vdash \Gamma_i$ is a derivation of σ, and we are done.

Definition 4. *We say that t is only sent under one-way functions in the runs of a process L if there is a one-way position set Π such that $t \in \mathsf{par}(\mathcal{T}_L, \Pi)$, where $\mathcal{T}_L$ is the set of terms of L.*

A term t is protected in L if it is only sent under one-way functions, or not at all. The set of the terms protected in L is thus

$$\mathsf{prot}_L(\Theta, \Pi) = \mathsf{par}_L(\mathcal{T}_L, \Pi) \cup (\mathcal{T} \setminus \mathcal{T}_L^{\sqsupseteq})$$

Proposition 1. *If a term t is only sent under one-way functions in the runs of a process L, then t cannot be derived by the observers of these runs. Formally,*

$$\mathsf{Onwy}(\Pi) \wedge t \in \mathsf{par}_L(\Gamma_X, \Pi) \Longrightarrow \Gamma_X \nvdash t$$

Proof. Suppose that t is derivable from the terms Θ in a run. By the definition of Onwy, $\mathsf{pur}_t(\Theta, \Pi) \vdash t$. But by assumption, t does not appear in $\mathsf{pur}_t(\Theta, \Pi)$. Since t is an atomic term, that means that t cannot be derived from $\mathsf{pur}_t(\Theta, \Pi)$, and so it cannot be derived from Θ.

Corollary 1. *If a term t is protected in a process L, and in the initial environments of all principals, then no run of L will make it computable for any principal for which it was not computable initially, before any runs of L, i.e.*

$$\mathsf{Onwy}(\Pi) \ \wedge \ t \in \mathsf{prot}_L(\mathcal{T}_L, \Pi) \ \wedge \ \forall X \in \mathcal{W}.\ t \in \mathsf{prot}_L(\Gamma_X^\iota, \Pi) \implies \mathsf{Only}(t; G_t)$$

where $G_t = \{X \in \mathcal{W} \mid \Gamma_X^\iota \vdash t\}$.

Proposition 2. *Let L be a process, and let k be an atomic term which only appears as the first argument of E in any term in the runs of L. Then, k cannot be derived by the observers of these runs.*

Proof. Since k is atomic, and the $\vdash$ relation introduces no new variables on the right-hand side, the fact that k does not appear in any element of $\mathsf{pur}_k(\mathcal{T}_L, \langle E, 1\rangle)$ implies $\mathsf{pur}_k(\mathcal{T}_L, \langle E, 1\rangle) \nvdash k$. Since $\langle E, 1\rangle$ is one-way (by lemma 3), definition 3 yields $\Theta \nvdash k$.

Remark. If $\Theta = \{E(Fkm)n\}$, and $\mathsf{Onwy}(E, 1)$, then $\Theta \nvdash Fkm$. Note, however that this does not imply $\Theta \nvdash k$, since the algebraic theory may include, e.g. the equation $E(Fxy)z = x$. However, if $\mathsf{Onwy}\{\langle E, 1\rangle, \langle F, 1\rangle\}$, then $\Theta \nvdash k$ can indeed be proven.

4.2 Guarding by Trapdoor Functions

Definition 5. *Given sets $\Upsilon, \Theta \subseteq \mathcal{T}$ and $\Psi \subseteq \Sigma \times \mathbb{N} \times \mathbb{N}$, with the projections $\Psi_0 = \{\langle f, i\rangle \mid \exists k.\ \langle f, i, k\rangle \in \Psi\}$ and $\Psi_1 = \{\langle f, k\rangle \mid \exists i.\ \langle f, i, k\rangle \in \Psi\}$, then the operator*

$$\begin{aligned}\mathsf{pad}_\Upsilon(\Theta, \Psi) \quad = \quad \big\{\langle \sigma, \kappa\rangle \in \Upsilon^2 \setminus \Theta^2 \ \mid \ \forall t \in \Theta \forall s \sqsubseteq t \forall f \in \Sigma \forall i \in \mathbb{N}.\ \big(s = f(\sigma)_i \\ \Rightarrow \exists k \in \mathbb{N}.\ \langle f, i, k\rangle \in \Psi \ \wedge \ s = f(\kappa)_k\big)\big\}\end{aligned}$$

extracts just those pairs of terms $\langle \sigma, \kappa\rangle \in \Upsilon^2$ where σ only occurs in a Ψ_0-position, if κ occurs in a Ψ_1-position within the same subterm.

Like before, $\mathsf{pad}(\Theta, \Psi)$ stands for $\mathsf{pad}_{\mathcal{T}}(\Theta, \Psi)$. We further define

$$\mathsf{msg}_{\mathcal{T}}(\Theta, \Psi) = \mathsf{par}_{\mathcal{T}}(\Theta, \Psi_0) \qquad \mathsf{key}_{\mathcal{T}}(\Theta, \Psi)\sigma = \{\kappa \mid \langle \sigma, \kappa \rangle \in \mathsf{pad}_{\mathcal{T}}(\Theta, \Psi)\}$$

Definition 6. *We say that $\Psi \subseteq \Sigma \times \mathbb{N} \times \mathbb{N}$ is a trapdoor set if for every $\langle f, i, j \rangle \in \Psi$ holds that whenever an $\langle f, i \rangle$-subterm σ can be extracted from $f(\sigma)_i$, then some $\langle f, j \rangle$-subterm κ must also be computable. Formally,*

$$\mathsf{Trap}(\Psi) = \forall \Theta\ \forall \sigma \in \mathsf{msg}(\Theta, \Psi).\ \Theta \vdash \sigma \implies \exists \kappa \in \mathsf{key}(\Theta, \Psi)\sigma.\ \Theta \vdash \kappa$$

We abbreviate $\mathsf{Trap}(\{\langle f, i, k \rangle\})$ to $\mathsf{Trap}(f, i, k)$, and for n-ary f write $\mathsf{Trap}(f, k)$ instead of $\mathsf{Trap}(\{\langle f, 1, k \rangle, \ldots, \langle f, n, k \rangle\})$.

Proposition 3. *The function E from sec. 3.2 satisfies $\mathsf{Trap}(E, 2, 1)$.*

Proof. Let Θ be a set of terms, $\Pi = \langle E, 1 \rangle$, and $\sigma \in \mathsf{msg}(\Theta, \Psi)$ such that $\Theta \vdash \sigma$ but $\Theta \nvdash g$ for any $g \in \mathsf{key}(\Theta, \Psi)$. Let $\mathcal{T}$ be a sequence $T_1 \vdash \ldots \vdash T_n \vdash \sigma$ where $T_1 \subseteq \Theta$ and each $\vdash$ step consists of a single application of a rewrite rule. Then some T_i must be of the form $S \cup \{k, Eky\} \vdash S \cup \{k, Eky, y\}$ where $\sigma \sqsubseteq y$. Hence $\Theta \vdash k$ and $k \in \mathsf{key}(\Theta, \Psi)$, contradicting our assumption.

The following three results follow directly from the definitions, so we omit their proofs.

Lemma 4. *For every Θ and a trapdoor set Ψ holds*

$$\{\mathsf{key}(\Theta, \Psi)\sigma \mid \sigma \in \mathsf{msg}(\Theta, \Psi)\}\ \ \mathsf{guards}_{\Theta}\ \ \mathsf{msg}(\Theta, \Psi)$$

Proposition 4. *If a term σ is only sent under trapdoor functions in the runs of a process L, then every computation leading to σ must pass through a trapdoor κ. Formally,*

$$\mathsf{Trap}(\Psi)\ \wedge\ \sigma \in \mathsf{msg}(\mathcal{T}, \Psi) \Longrightarrow \mathsf{key}(\mathcal{T}, \Psi)\sigma \neq \emptyset.$$

Corollary 2. *If a term σ is only sent under trapdoor functions in the runs of a process L, and if all of its trapdoors are contained in Γ, then σ is guarded by Γ, i.e.*

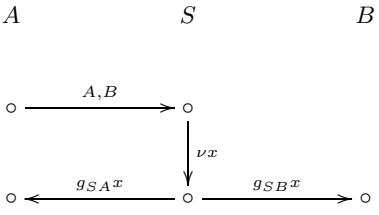
$$\mathsf{Trap}(\Psi)\ \wedge\ \sigma \in \mathsf{msg}(\mathcal{T}_L, \Psi)\ \wedge\ \mathsf{key}(\mathcal{T}_L, \Psi)\sigma \subseteq \Gamma \Longrightarrow \{\Gamma\}\ \mathsf{guards}_L\ \sigma$$

Remark. One-way functions implement a simple form of guarding, where s is guarded by the function symbol f in the term fs. Trapdoor functions implement guarding where s can be extracted from a term fst only if t is known. Equations between terms lead to more complicated forms of guarding. E.g., if $g^{\wedge}(-)$ is a one-way function satisfying $(g^{\wedge}y)^{\wedge}z = (g^{\wedge}z)^{\wedge}y$, then z cannot be extracted from $(g^{\wedge}y)^{\wedge}z$, but this term can be computed without knowing z, from $\{g^{\wedge}z, y\}$.

5 Secrecy Derivations

5.1 Key Distribution

Consider the abstract key distribution protocol $\mathsf{KD}[A, B, S]$



The principal A here requests a key to communicate with B and the server S generates a fresh key and distributes it, guarded by the functions g_{SA} and g_{SB}. The desired secrecy property is that at the final state of KD, only A,B and S have x, provided that they are honest.

To assure this, we impose the following axioms

$$\forall Y \in \mathcal{W}.\ \mathsf{Secr}^{\iota}(g_{SY}, g_{SY}^{-1}; S, Y) \qquad (\mathsf{sg})$$

$$\mathsf{Auth}^{\mathsf{KD}}(g_{SA}, g_{SA}^{-1}, g_{SB}, g_{SB}^{-1}; A, B, S) \qquad (\mathsf{ag})$$

$$\mathsf{Auth}^{\mathsf{KD}}(x; A, B, S) \qquad (\mathsf{ax})$$

which say (sg) that at the *initial* state ι, only S and Y can construct and remove g_{SY}, and (ag,ax) that A, B and S can each ascertain that there are no undesired flows involving g's and x. The desired secrecy property is now obtained using the secrecy rule

$$\frac{\dfrac{(\mathsf{sg}) \wedge (\mathsf{ag})}{\mathsf{Secr}^{\mathsf{KD}}(g_{SY}^{-1}; S, Y)\ \big|_{Y\in\{A,B\}}} \qquad \dfrac{g_{SY}x \in \Gamma^{\mathsf{KD}}_{S,Y}\ \big|_{Y\in\{A,B\}}}{g_{SX}^{-1} \vdash_{S,Y} x\ \big|_{Y\in\{A,B\}}} \qquad \dfrac{(\mathsf{ax}) \wedge \mathsf{H}(A, B, S)}{\{\{g_{SA}^{-1}\}, \{g_{SB}^{-1}\}\}\ \mathsf{guards}\ x}}{\mathsf{Secr}^{\mathsf{KD}}(x; A, B, S)}$$

where $\mathsf{H}(A, B, S)$ is the usual honesty assumption, that principals act according to the protocol. To see how $(\mathsf{ax})\wedge\mathsf{H}(A, B, S)$ implies that $\{\{g_{SA}^{-1}\}, \{g_{SB}^{-1}\}\}$ guards x, recall that $\mathsf{Auth}(x; A, B, S)$ means that $(\overline{L}_A \restriction x) \approx (\overline{L}_S \restriction x) \approx (\overline{L}_B \restriction x)$, at the final state of KD. In other words, all three principals A, B and S can in each run of KD establish the same complete view of all of their actions with the terms containing x. Of course, each of them observes only her own actions, but the assumption $\mathsf{Auth}(x; A, B, S)$ asserts that this suffices to allow each of them to also prove the exact order of actions of the other two. In particular, A and B can both prove that S has only sent $g_{SA}x$ and $g_{SB}x$, and nothing else.

In this reasoning, the authentication assumptions are encapsulated in axioms (ag) and (ax), which *postulate* that there are no runs viewed differently by different principals. This means that neither of them can be proven, nor needs to be proven, for the abstract protocol. They just specify requirements which the

implementations of the abstract protocol need to satisfy. This is analogous to the derivational approach to authentication, where the axiom (cr) specifies the required property of the challenge-response functions c and r, as an implementation task. The task here is to implement the guard function g, and discharge the open authenticity assumption. The abstract secrecy property of KD, proven above, must be preserved and realized through such implementations.

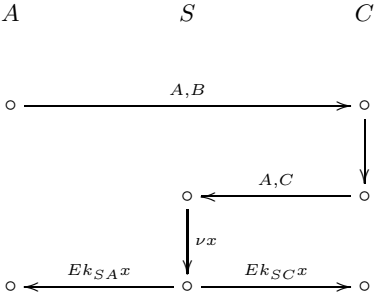
Refining KD. Suppose that we are given an encryption algorithm E, with the decryption algorithm D, and a symmetric key k_{SY} shared by the server S with every $Y \in \mathcal{W}$. Assume that these data satisfy the following axioms[5]

$$\mathsf{Onwy}\langle E, 1\rangle \qquad (\mathsf{oE})$$

$$\mathsf{Trap}\langle E, 2, 1\rangle \qquad (\mathsf{tE})$$

$$\forall Y \in \mathcal{W}.\mathsf{Secr}^{\iota}(k^{SY}; S, Y) \qquad (\mathsf{sk})$$

First Try. Setting $g_{SY}x = Ek_{SY}x$, one can easily derive (sg) from (sk), and the secrecy of k_{SY} follows from (sk), (oE), and $\mathsf{Auth}^{\mathsf{KD}}(k_{SY}; Y, S)$. However, (ag)[6] and (ax) are not satisfied, because $g_{SY} = Ek_{SY}$ allows runs which may leave each principal with a different view, e.g.



Here S does not know that A wanted to speak to B, A does not know that C participates the run, and B does not know that anything happened at all.

Second Try. The principal A needs to prove (ax) that S has only sent x in the two tokens, intended for A and B, guarded by g_{SA} or g_{SB}; and (ag) that the guards have not been compromised. Since A does not have g_{SB}^{-1}, she cannot check (ax) directly. So she must check it indirectly, relying for the authentications (ag) and (ax) on the information received from S. This idea is realized by adding peer's identity in the term $g_{SA}x$, constructed by S for A. Ditto for $g_{SB}x$.

Proposition 5. *The protocol* KD*, with axioms* (sg,ag,ax)*, can be refined by setting*

$$g_{SA}x = Ek_{SA}Bx \qquad g_{SB}x = Ek_{SB}Ax$$

[5] For simplicity, the term algebra is here untyped, and any term can be used as a key; typing can be introduced as needed.

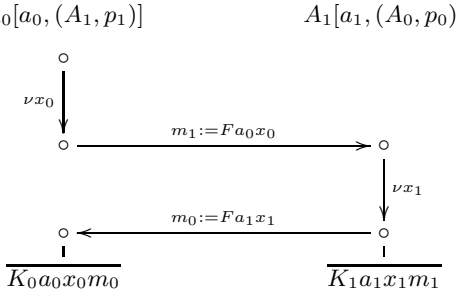
[6] Note that $\mathsf{Trap}\langle E, 2, 1\rangle$ implies $\mathsf{Auth}(Ek, Ek^{-1}) \iff \mathsf{Auth}(k)$.

provided that (oE) $\mathsf{Onwy}\langle E, 1\rangle$, (tE) $\mathsf{Trap}\langle E, 2, 1\rangle$, *and* (sk) $\mathsf{Secr}^{\iota}(k^{SY}; S, Y), Y \in \{A, B\}$ *are satisfied. Under these assumptions, the proof of* KD*'s secrecy property specializes to its refinement.*

The validity of (sg) follows as in the first try. The validity of (ag) and (ax) follows from the honesty assumption: if A receives $Ek_{SA}Bx$, then the presence of B in that message means the only other token that the honest server S would send is $Ek_{SB}Ax$.

5.2 Key Agreement

The generic key agreement protocol KA is based on abstract public key infrastructure: each principal $X \in \mathcal{W}$ is given a long term private key a_X, corresponding to a public key p_X, as well as an "address book" $\{(Y, p_Y)\}_{Y\in\mathcal{W}}$ of public keys of all principals. This means that the view Γ_X^{σ} for every σ and X contains $\{a_X, (Y, p_Y) \mid Y \in \mathcal{W}\}$. But the participants of the abstract key agreement protocol $\mathsf{KA}[A_0, A_1]$ only need to know each other's public key, so it becomes



Besides the displayed secret data, the operations F and K_i may also depend on the public data (but this clutters notation and plays no role in the reasoning). The following axioms are imposed:

$$\mathsf{Onwy}(F) \tag{oF}$$

$$\forall X \in \mathcal{W}.\ \mathsf{Secr}^{\iota}(a_X; X) \tag{sa}$$

$$\mathsf{Auth}^{\mathsf{KA}}(a_0, x_0, a_1, x_1; A_0, A_1) \tag{aax}$$

$$K_0a_0x_0(Fa_1x_1) \;=\; K_1a_1x_1(Fa_0x_0) \tag{agr}$$

$$\big\{\{a_0, x_0\}, \{a_1, x_1\}\big\}\ \mathsf{guards}\ \{K_0a_0x_0m_0, K_1a_1x_1m_1\} \tag{gua}$$

The secrecy rule now gives

$$\frac{\dfrac{(\mathsf{oF}) \wedge (\mathsf{sa}) \wedge (\mathsf{aax}) \wedge \mathsf{H}(A_0, A_1)}{\mathsf{Secr}^{\mathsf{KA}}(a_i, x_i; A_i)\ \big|_{i\in\{0,1\}}} \qquad \dfrac{m_i \in \Gamma_{A_i}^{\mathsf{KA}}\ \big|_{i\in\{0,1\}}}{a_i, x_i \vdash_{A_i} K_ia_ix_im_i\ \big|_{i\in\{0,1\}}}}{\mathsf{Secr}^{\mathsf{KA}}(K_0a_0x_0m_0, K_1a_1x_1m_1; A_0, A_1)}\ (\mathsf{gua})$$

It further follows from (aax) that the messages Fa_0x_0 and Fa_1x_1 are exchanged as desired and assigned to m_1 and m_0 respectively. Axiom (agr) then implies that at the final state of KA, the secret keys $K_0a_0x_0m_0$ and $K_1a_1x_1m_1$ agree, and yield the key $k = K_0a_0x_0m_0 = K_0a_1x_1m_1$. This is the desired outcome of the protocol.

Refining KA. The abstract KA-scheme subsumes the large family of protocols arising from Diffie and Hellman's paradigm [15]. It includes the MTI family and their descendants, UM, KEA, MQV and others. In combination with other security components, they are used in several widely used protocol suites [16,17].

The minimal algebraic structure needed for the DH-style key agreement is a multiplicative group $\mathbb{G}$ with a binary operation $(-^\wedge -) : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}$, satisfying

$$x^\wedge 1 = x \tag{eq1}$$

$$(x^\wedge y)^\wedge z = (x^\wedge z)^\wedge y \tag{eq2}$$

$$\mathsf{Onwy}(x^\wedge) \tag{dl}$$

$$\big\{\{x^\wedge y, z\}, \{x^\wedge z, y\}\big\} \ \mathsf{guards}_{\mathsf{KA}} \ \big\{(x^\wedge y)^\wedge z, (x^\wedge z)^\wedge y\big\} \tag{cdh}$$

Axiom (dl) is the abstract version of the Discrete Logarithm (DL) assumption, and (cdh) is related to the Computational Diffie-Hellman (CDH) assumption.

Besides the group $\mathbb{G}$, the assumed infrastructure includes a chosen element $g \in \mathbb{G}$, intended to generate a large subgroup of $\mathbb{G}$. For each principal $X \in \mathcal{W}$, the public key p_X, corresponding to the long-term private key a_X is now specified in the form $p_X = g^\wedge a_X$. So for all X and σ, the view Γ_X^σ now contains a_X, g and $\{(Y, g^\wedge a_Y)\}_{Y \in \mathcal{W}}$.

The DH-style key agreement protocols now fall into two subfamilies, depending on whether the key derivation functions K_0 and K_1 are the same or not.

Asymmetric Key Derivation. Given functions H_0 and H_1, that satisfy

$$H_0 a_0 x_0 (F a_1 x_1) \ = \ H_1 a_1 x_1 (F a_0 x_0) \tag{agrH}$$

$$\big\{\{a_0, x_0\}, \{a_1, x_1\}\big\} \ \mathsf{guards} \ \{H_0 a_0 x_0 (F a_1 x_1), H_1 a_1 x_1 (F a_0 x_0)\} \tag{guaH}$$

setting

$$K_0 axm = J(H_0 axm)(H_1 axm) \qquad K_1 axm = J(H_1 axm)(H_0 axm)$$

validates (agr) and (gua). This subsumes the general scheme of the first two MTI protocols [18]

protocol	Fax	Jxy	$H_0 axm$	$H_1 axm$
MTI/A(f)	$g^\wedge(fax)$	$x \cdot y$	$p^\wedge(fax)$	$m^\wedge a$
MTI/B(f)	$p^\wedge(fax)$	$x \cdot y$	$g^\wedge(fax)$	$m^\wedge(1/a)$

The variable p still denotes peer's public key. When proving (agr), substitute each p_i by $g^\wedge a_i$. The original MTI protocols are obtained by setting $fxy = x^i \cdot y$. For $i = 1$, replacing the function $Jxy = xy$ in MTI/A by $Jxy = x + y$ and extending the group axioms with ring axioms yields the core of the KEA protocol; taking Jxy to append and then hash x and y yields the core of the "Unified Model" protocol (UM). The security properties gained by these variations, discussed in [19,17], are beyond the scope of our current axioms, but can be captured in refinements.

Symmetric Key Derivation. Given a function K, satisfying

$$Ka_0x_0(Fa_1x_1) \;=\; Ka_1x_1(Fa_0x_0) \qquad \text{(agrK)}$$

$$\{\{a_0, x_0\}, \{a_1, x_1\}\} \text{ guards } \{Ka_0x_0(Fa_1x_1), Ka_1x_1(Fa_0x_0)\} \qquad \text{(guaK)}$$

can, of course, implement both K_0 and K_1 in the protocol KA. This subsumes the third MTI protocol, and the scheme which we denote MQV/D.

protocol	Fax	$Kaxm$
MTI/C(f)	$p^\wedge(fax)$	$(m^\wedge(1/a))^\wedge(fax)$
MQV/D(f)	$g^\wedge(fax)$	$(Rpm)^\wedge(ra(fax))$

where R and r are required to satisfy

$$x^\wedge(ryz) = R(x^\wedge y)(x^\wedge z) \qquad \text{(agrR)}$$

$$\{a, x\} \text{ guards } ra(fax) \qquad \text{(guar)}$$

Instantiating to $Rxy = (x^\wedge y) \cdot y$ and $rxy = x \cdot (g^\wedge y) + y$ validates these axioms and gives the MQV protocol [20,21], but there are other interesting choices. Instantiating $Rxy = rxy = fxy = y$ yields the DH protocol, which, of course, does not validate (guar).

Proposition 6. *The protocol* KA*, with axioms* (oF,sa,aax,agr,gua)*, can be refined as above, using the asymmetric key derivation functions H_0, H_1, or the symmetric key derivation scheme K, provided that axioms* (oF,sa,aax,agrX,guaX) *hold for* X $\in \{H, K\}$.

Using the algebraic structure satisfying (eq1,eq2,cdh,dl)*, the protocols* MTI/A, MTI/B,UM *and* KEA *can be obtained as further refinements of the asymmetric scheme, whereas the protocols* MTI/C *and* MQV/D *are further refinements of the symmetric scheme, assuming, in the latter case, also* (agrR,guar).

The generic secrecy property, proven for the protocol KD*, is inherited by all of its refinements.*

6 Conclusions

The main contribution of this work is an extension of the framework for composing and refining security protocols, that allows proofs of the secrecy properties realized by the protocols. The secrecy concepts are defined in terms of the abstract computability relation, and the secrecy derivations are built from the rules derivable from the minimal assumptions about this relation. The secrecy properties are derived from the axioms attached to the basic protocol components. The axioms specify the requirements/assumptions that need to be discharged through refinement and implementation of abstract operations. Some axioms encapsulate the authenticity assumptions. Since they embody a different, logically dual concern from secrecy, these assumptions are cumbersome to realize concurrently with it. Encapsulating them in some cases significantly simplifies

the secrecy proofs. The genericity of the approach allows reusable treatment of broad families of related protocols. All derivations are stored, and were originally constructed, in the prototype version of a software tool, which is freely available for download [3]. The structure of the presented modeling methodology is very much influenced by its role of the semantical underpinning of this tool.

There are also interesting further directions to be explored. Although our initial explorations in this area are based on a symbolic model, there is no need to limit ourselves in this direction. Indeed, we could develop different secrecy logics with different semantics based on, for example, information-theoretic or computational aspects of cryptography, depending on the types of cryptosystems being used. The ability to derive and employ different secrecy logics would allow us to develop a pluggable semantics for cryptographic protocol analysis that would allow us to reason over multiple domains.

Acknowledgements. We thank Carolyn Talcott for careful reading and many useful comments.

References

1. Cervesato, I., Meadows, C., Pavlovic, D.: An encapsulated authentication logic for reasoning about key distribution protocols. In Guttman, J., ed.: Proceedings of CSFW 2005, IEEE (2005) 48–61
2. Anlauff, M., Pavlovic, D., Waldinger, R., Westfold, S.: Proving authentication properties in the Protocol Derivation Assistant. In: Proceedings of ARSPA 2006. Lecture Notes in Computer Science, IEEE (2006) to appear.
3. Anlauff, M., Pavlovic, D.: Pda download web site. `http://www.kestrel.edu/software/pda` (2003–6)
4. Durgin, N., Mitchell, J., Pavlovic, D.: A compositional logic for proving security properties of protocols. J. of Comp. Security **11**(4) (2004) 677–721
5. Datta, A., Derek, A., Mitchell, J., Pavlovic, D.: A derivation system and compositional logic for security protocols. J. of Comp. Security **13** (2005) 423–482
6. Meadows, C., Pavlovic, D.: Deriving, attacking and defending the GDOI protocol. In Ryan, P., Samarati, P., Gollmann, D., Molva, R., eds.: Proc. ESORICS 2004. Volume 3193 of Lecture Notes in Computer Science., Springer Verlag (2004) 53–72
7. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and Authenticated Key Exchanges. Designs, Codes, and Cryptography **2** (1992) 107–125
8. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Proc. CRYPTO '93, Springer-Verlag (1994) 232–249
9. Pratt, V.: Modelling concurrency with partial orders. Internat. J. Parallel Programming **15** (1987) 33–71
10. Fabrega, F.J.T., Herzog, J., Guttman, J.: Strand spaces: What makes a security protocol correct? Journal of Computer Security **7** (1999) 191–230
11. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). J. of Cryptology **15**(2) (2002) 103–127
12. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6** (1998) 85–128
13. Millen, J.: On the freedom of decryption. Information Processing Letters **86**(6) (2003) 329–333

14. Goldreich, O.: Foundations of Cryptography. Volume I: Basic Tools. Cambridge University Press (2000)
15. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**(6) (1976) 644–654
16. Kaufman, C., Perlman, R., Speciner, M.: Network Security. Private Communication in a Public World. 2 edn. Computer Networking and Distributed System. Prentice Hall PTR (2002)
17. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment. Information Security and Cryptography. Springer-Verlag (2003)
18. Matsumoto, T., Takashima, Y., Imai, H.: On seeking smart public-key distribution systems. Transactions of the IECE (Japan) **69** (1986) 99–106
19. Blake-Wilson, S., Menezes, A.: Authenticated Diffie-Hellman key agreement protocols. In: SAC '98: Proceedings of the Selected Areas in Cryptography, London, UK, Springer-Verlag (1999) 339–361
20. Menezes, A., Qu, M., Vanstone, S.: Some new key agreement protocols providing mutual implicit authentication. In: SAC '95: Proceedings of the Selected Areas in Cryptography, London, UK, Springer-Verlag (1995) 22–32
21. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Technical Report CORR 98-05, University of Waterloo (1998) also in [22].
22. P1363 Working Group: The IEEE P1363 home page. standard specifications for public-key cryptography. `http://grouper.ieee.org/groups/1363/` (2005)
23. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems **21**(4) (1993) 706–734
24. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in distributed systems: theory and practice. ACM Trans. on Comput. Syst. **10**(4) (1992) 265–310
25. Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Transactions on Software Engineering and Methodology **9**(4) (2000) 410–442
26. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565

Appendix: Protocol Derivation System

Syntactic Categories: Conceptually, authenticity and secrecy are dual, in the sense that authenticity says that some principals know something, while secrecy says that that they do not know something else. Formally, though, both secrecy and authenticity can be analyzed in terms of partial ordering: principals authenticate each other by establishing the same order of their joint actions, whereas the failures of secrecy can be viewed as connecting some data that are known, and some that should not be known through a relation of computability, which is transitive and reflexive. So we reason over several partially ordered syntactic categories:

- *data* are represented as terms from an algebra $\mathcal{T}$, given with an abstract subterm relation $\sqsubseteq$;
  - to support information-theoretic security analyses, $\mathcal{T}$ must be given with a frequency distribution $\mathsf{Prob} : \mathcal{T} \longrightarrow [0, 1]$;

  - to support computational security analyses, besides the frequency distribution, $\mathcal{T}$ must be given with a representation $\mathcal{T} \longrightarrow \Sigma^*$ of terms as strings (usually from the alphabet $\Sigma = \{0, 1\}$), available for processing e.g. by turing machines;
- *principals* are collected in the set $\mathcal{W}$ ordered by the relation $\Subset$, which may be interpreted as "speaks for" [23,24] or "acts for" [25], etc.
- *actions* are generated from terms and principals by the constructors including

action	constructor	form	meaning
send	$\mathcal{T} \times \mathcal{W}^2 \overset{\langle\rangle}{\hookrightarrow} \mathcal{A}$	$\langle t : A \to B \rangle$	the term t is sent, purportedly from A to B
receive	$\mathsf{Var}_{\mathcal{T}} \times \mathsf{Var}_{\mathcal{W}}^2 \overset{()}{\hookrightarrow} \mathcal{A}$	$(x : Y \to Z)$	a term, source and destination are received into the variables x, Y, and Z
match	$\mathcal{T} \times \Sigma_{\mathcal{T}} \times \mathsf{Var}_{\mathcal{W}} \overset{(/)}{\hookrightarrow} \mathcal{A}$	$(t/p(x))$	the term t is matched with the pattern $p(x)$
new	$\mathsf{Var}_{\mathcal{T}} \overset{(\nu)}{\hookrightarrow} \mathcal{A}$	(νx)	a fresh value is created and stored in the variable x

We often use partial descriptions, and elide e.g. the source and the destination of a message, as in $\langle t \rangle$, or (y). For simplicity, we also omit the explicit type structure. However, all term variables are assumed to be local to a principal, and thus come with a map $\mathsf{Var}_{\mathcal{T}} \longrightarrow \mathcal{W}$; each principal is assumed to have an infinite supply of local variables.

Execution Model: Processes are represented as partially ordered multisets (pomsets [9]) of actions attributed to principals. More precisely, a **process** is an assignment $\mathbb{L} \xrightarrow{L} \mathcal{A} \times \mathcal{W}$ such that (a) $(\mathbb{L}, <)$ is a well-founded partial order, and (b) $p < q$ implies $L_{\mathcal{W}}(p) \Subset L_{\mathcal{W}}(q)$ or $L_{\mathcal{W}}(p) \ni L_{\mathcal{W}}(q)$. We abuse notation and write an action $p \in \mathbb{L}$ such that $L_{\mathcal{A}} p = a$ and $L_{\mathcal{W}} p = A$ as $a_A \in \mathbb{L}$, or even a, although, of course, several elements of $\mathbb{L}$ may correspond to the same action by the same principal.

A **run** ℓ extends a process L by a choice of *communication links*, which assign to each receive action a unique send action. Formally, a run is thus a pair

$$\ell \;=\; \langle L,\;\; \surd :\; \mathsf{recvs}(L) \longrightarrow \mathsf{sends}(L) \rangle \qquad (x : Y \to Z)_A \mapsto \langle t : S \to R \rangle_B$$

such that $\surd(x) \not> (x)$. A run ℓ thus induces the partially ordered set $\mathbb{L}^\ell$, obtained by extending the ordering of $\mathbb{L}$ by $\surd(x) < (x)$. One can thus think of a run as the pomset extension $\mathbb{L} \hookrightarrow \mathbb{L}^\ell$, where each receive action (x) from $\mathbb{L}$ has in $\mathbb{L}^\ell$ a chosen predecessor $\langle t \rangle = \surd(x)$. This is, of course, just another formalization of Lamport's ordering of actions [26]. The resulting model is also similar, and has been influenced by strand spaces and bundles [10].

The actions in a run $\ell = \langle \mathbb{L}, \surd \rangle$ are then executed in order: each $a \in \mathbb{L}$ can be executed only after all $b < a$ have been executed. Executing an action changes state. A **state** of a run ℓ is a triple $\sigma = \langle \sigma^1, \sigma^2, \sigma^3 \rangle$, where

- $\sigma^1 \supset \mathbb{L}^\ell$ is a poset, extending each maximal chain of $\mathbb{L}^\ell$ by exactly one element, denoted $\blacktriangledown$, that marks the execution point;

- $\sigma^2 = \{x_1, \ldots, x_m\}$ is a finite set of the variables bound to *freshly generated* nonces or keys,
- $\sigma^3 = \{y_1 := t_1, \ldots y_n := t_n\}$ is a finite set of the assignments, that result from the executed actions; formally, it can be viewed as a partial map from the variables y_i to the terms t_i, such that its domain $\{y_1 \ldots y_n\}$ is disjoint from $\sigma^2 = \{x_1 \ldots x_m\}$.

As always, the variables are taken up to renaming (i.e. modulo α-conversion).

At the *initial state* ι of every run ℓ, all the markers in ι_1 are set below the minimal elements of $\mathbb{L}^\ell$, and $\iota_2 = \iota_3 = \emptyset$.If the execution of the run ℓ has reached a state σ, the transition $\sigma \longrightarrow \tau$ proceeds as follows[7]:

action	if in σ^1	and if	then set τ^3 to	set τ^2 to	and in τ^1
send	$^{\blacktriangledown}\langle t\rangle_C$	$FV(t) \subseteq \sigma^2$	σ^3	σ^2	$\langle t\rangle_C^{\blacktriangledown}$
receive	$^{\blacktriangledown}(x)_D$	$\surd(x)_D = \langle t\rangle_C$ $\wedge\ \ x \notin \sigma^2$	$\sigma^3 \cup \{x := t\}$	σ^2	$(x)_D^{\blacktriangledown}$
match	$^{\blacktriangledown}(t/p(x))_D$	$t = p(u)$ $\wedge\ \ x \notin \sigma^2$	$\sigma^3 \cup \{x := t\}$	σ^2	$(t/p(x))_D^{\blacktriangledown}$
new	$^{\blacktriangledown}(\nu x)_D$	$x \notin \sigma^2$	σ^3	$\sigma^2 \cup \{x\}$	$(\nu x)_D^{\blacktriangledown}$

The local state σ_A is the part of the state σ that can be observed by the principal A. Its components are thus

- the poset $\sigma^1_A \subseteq \sigma^1$, spanned[8] by the actions of A;
- the set σ^2_A obtained by restricting σ^2 to A's local variables, and
- the partial map σ^3_A, obtained by restricting σ^3 to A's local variables.

A **protocol** is a specification of a process and a nonempty set of desired runs. The participants of a protocol are usually called *roles*, and are denoted by the variables for principals. Since the pomset representing a run extends the pomset representing a process, and the restriction of the run to the process is usually obvious, a protocol is usually specified just by its desired runs. Such a specification should thus be construed as a proof task: show that the principals can prove that the run which they participated is as desired.

[7] For compactness of the table, we omit the source and the destination fields, which are just passed from the received message to the receiving variables.

[8] Just like σ^1 is a $\blacktriangledown$-marking of $\mathbb{L}^\ell$, σ^1_A is a $\blacktriangledown$-marking of $\mathbb{L}^\ell_A = \{\xi \in \mathbb{L}^\ell \mid L_{\mathcal{W}}\xi = A\}$.