Security Analysis of Cryptographically Controlled Access to XML Documents

Martín Abadi
Computer Science Department
University of California at Santa Cruz
abadi@cs.ucsc.edu

Bogdan Warinschi[*]
Computer Science Department
Stanford University
bogdan@theory.stanford.edu

ABSTRACT

Some promising recent schemes for XML access control employ encryption for implementing security policies on published data, avoiding data duplication. In this paper we study one such scheme, due to Miklau and Suciu. That scheme was introduced with some intuitive explanations and goals, but without precise definitions and guarantees for the use of cryptography (specifically, symmetric encryption and secret sharing). We bridge this gap in the present work. We analyze the scheme in the context of the rigorous models of modern cryptography. We obtain formal results in simple, symbolic terms close to the vocabulary of Miklau and Suciu. We also obtain more detailed computational results that establish security against probabilistic polynomial-time adversaries. Our approach, which relates these two layers of the analysis, continues a recent thrust in security research and may be applicable to a broad class of systems that rely on cryptographic data protection.

1. INTRODUCTION

A classic method for enforcing policies on access to data is to keep all data in trusted servers and to rely on these servers for mediating all requests by clients, authenticating the clients and performing any necessary checks. An alternative method, which is sometimes more attractive, consists in publishing the data in such a way that each client can see only the appropriate parts. In a naive scheme, many sanitized versions of the data would be produced, each corresponding to a partial view suitable for distribution to a subset of the clients. This naive scheme is impractical in general. Accordingly, there has been much interest in more elaborate and useful schemes for fine-grained control on access to published documents, particularly for XML documents [4, 5, 7, 8, 14, 19, 23]. This line of research has led to efficient and elegant publication techniques that avoid data duplication by relying on cryptography. For instance, using those techniques, medical records may be published as XML documents, with parts encrypted in such a way that only the appropriate users (physicians, nurses, researchers, administrators, and patients) can see their contents.

The work of Miklau and Suciu [19] is a crisp, compelling example of this line of research. They develop a policy query language for specifying fine-grained access policies on XML documents and a logical model based on the concept of "protection". They also show how to translate consistent policies into protections, and how to implement protections by XML encryption [10]. Roughly, a protection is an XML tree in which nodes are guarded by positive boolean formulas over a set of symbols $\{K_1, K_2, \ldots\}$ that stand for cryptographic keys. Protections have a simple and clear intended semantics: access to the information contained in a node is conditioned on possession of a combination of keys that satisfies the formula that guards the node. For example, access to a node guarded by $(K_1 \wedge K_2) \vee K_3$ requires possessing either keys K_1 and K_2 or key K_3. (See Gifford's work for some of the roots of this approach [11].) Formally, a protection describes a function that maps each possible set of keys to the set of nodes that can be accessed using those keys, treating the keys as symbols. On the other hand, the use of keys for deriving a partially encrypted document is not symbolic: this process includes replacing the symbols $K_1, K_2, \ldots$ with actual keys, and applying a symmetric encryption algorithm repeatedly, bottom-up, to the XML document in question.

While Miklau and Suciu provide a thorough analysis of the translation of policies into protections, they leave a large gap between the abstract semantics of protections and the use of actual keys and encryption. The existence of this gap should not surprise us: an analogous gap existed in protocol analysis for 20 years, until recent efforts to bridge it [1, 2, 13, 15, 18, 20]. Concretely, the gap means that the protection semantics leaves many problematic issues unresolved. We describe two such issues, as examples:

- Partial information: It is conceivable that even when a node should be hidden according to a protection, the partially encrypted document may in fact leak some information about the data in that node.
- Encryption cycles: From the point of view of the abstract semantics, encryption cycles (such as encrypting

[*]This work was partly carried out while this author was affiliated with the University of California at Santa Cruz.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.

> a key with itself) are legitimate and do not contradict security. On the other hand, there are encryption algorithms that satisfy standard cryptographic definitions of security but that leak keys when encryption cycles are created.

More generally, there are many encryption methods and many notions of security for them (e.g., [3, 9, 12]), and it is not clear which one, if any, provides adequate guarantees for this application—nor is it exactly clear what those guarantees might be.

The immediate goal of this work is to bridge this gap by reconciling the abstract semantics of protections with a more concrete, computational treatment of security, and to define and establish precise security guarantees. We do not wish to replace the abstract semantics, which certainly has its place, but rather to complement it.

From a broader perspective, our goal is to develop, apply, and promote useful concepts and tools for security analysis in the field of database theory. These concepts and tools do not pertain to statistical techniques, which have long been known in database research (e.g., [6, 22]), but rather to cryptography. While sophisticated uses of cryptology in database research may have been of modest scope, there is an obvious need for database security, and we believe that cryptology has much to offer. In research on cryptographic protocols, formal and complexity-theoretic methods have been successful in providing detailed models and in enabling security proofs (sometimes automated ones). The same methods are beneficial for a broad class of systems that require security. Each application, however, can necessitate non-trivial, specific insights and results. In the techniques that we study, partial and multiple encryptions occur in (large, XML) data instances; we therefore depart from the situations most typically considered in the cryptography literature, towards data management. It is this specificity that motivates the present paper.

Overview of results

Our analysis is directed at the core of the framework of Miklau and Suciu, which aims to ensure data protection by an interesting combination of encryption schemes and secret sharing schemes [21]. As a formal counterpart to their loose, informal concept of data secrecy, we introduce a strong, precise cryptographic definition. The definition goes roughly as follows. Consider a protection for an XML document. An adversary is given an arbitrary set of keys, and the liberty of selecting two instantiations for the data in all nodes that occur in the XML document. The only restriction on these instantiations is that they should coincide on the nodes to which the adversary rightfully has access according to its keys and the abstract semantics of protections. In other words, the adversary selects two documents that contain the same information in the nodes it can access but may differ elsewhere. Then the adversary is given the partially encrypted document that corresponds to one of its two documents, and its goal is to decide which of the two instantiations was used in generating this partially encrypted document. Security means that the adversary cannot do much better than picking at random. It implies that the partially encrypted document reveals no information on the data in the nodes that should be hidden from the adversary, for otherwise this information would be sufficient to determine which instantiation was used.

Technically, we adapt and extend the approach of Abadi and Rogaway [1]. The novelties of this paper include the application to document access control, significant differences in basic definitions motivated by this application, and the treatment of secret sharing. First we provide an intermediate symbolic language for cryptographic expressions. We then define patterns of expressions; intuitively, a pattern represents the information that an expression reveals to an adversary. We show how to transform protections into cryptographic expressions, and use patterns to provide an equivalent semantics for protections. This equivalence is captured in Theorem 1. Going further, we relate expressions to concrete computations on bit-strings. The most difficult result of this paper is Theorem 2. Informally, it states that patterns faithfully represent the information that expressions reveal, even when expressions and patterns are implemented with actual encryption schemes (not symbolically). More precisely, we associate probability distributions with an expression and its pattern by mapping symbols to bit-strings and implementing encryption with a semantically secure encryption scheme [12], and prove that these distributions cannot be distinguished by any probabilistic polynomial-time algorithm. Our main theorem, Theorem 3, reconciles the abstract semantics of protections with the actual use of encryption. We establish that if data is hidden according to a protection, then it is secret according to our definition of secrecy.

Contents

The next section, Section 2, is mostly a review. In Section 3 we introduce our formal language for representing cryptographic expressions and give an alternative semantics to XML protections. Our main results are in Section 4: we give concrete interpretations to expressions and relate the formal semantics of protections to a strong definition of secrecy. We conclude in Section 5.

2. CONTROLLING ACCESS TO XML DOCUMENTS WITH PROTECTIONS

In this section we briefly recall the key aspects of the work of Miklau and Suciu. We focus on protections. We describe the derivation of partially encrypted documents from protections in the next section. We omit the policy query language because, for our purposes, we can discuss the protections generated from policies rather than the policies themselves.

We model XML documents as trees labeled with elements from a set $\mathsf{Data} = \{D_1, D_2, \dots\}$, which we tacitly assume to represent XML element names and actual data values, though we make no syntactic distinction between these two possibilities. We use pre-order representations for XML trees, described by the grammar:

$$\mathsf{XML} ::= (\mathsf{Data}) \mid (\mathsf{Data}, \mathsf{XML}, \mathsf{XML}, \dots, \mathsf{XML})$$

with terminals in $\mathsf{Data} \cup \{``(", ``)"\}$.

A *protection* consists of two components: a metadata XML tree obtained from an XML tree by adding metadata nodes, and a mapping that attaches to each node a positive boolean formula over $\mathsf{Keys} = \{K_1, K_2, \dots\}$. Metadata nodes have special kinds of labels, and it is assumed that they can be distinguished from the standard XML nodes. Their labels are one of the symbols OR and AND, or hold keys in

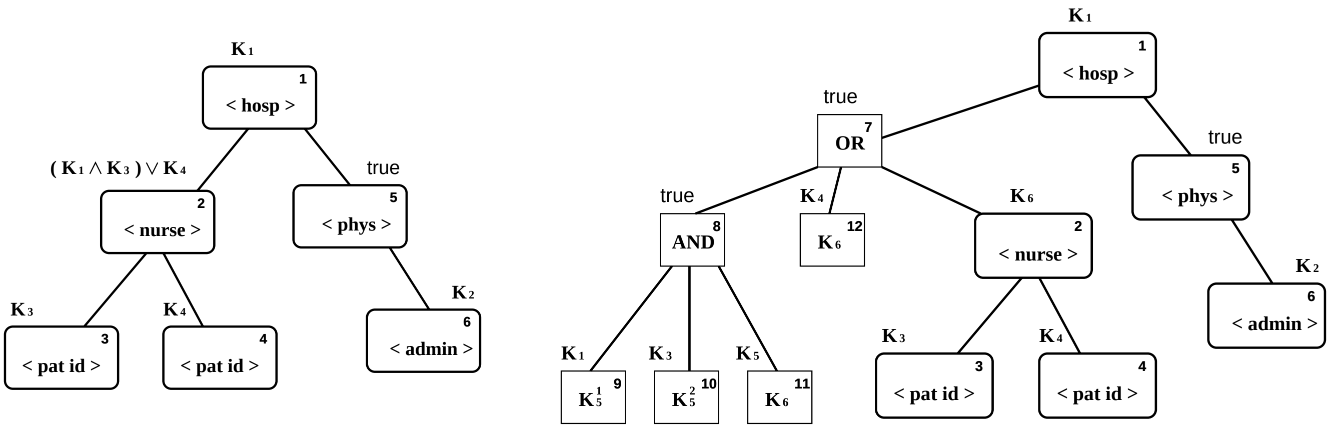

Figure 1: A tree protection (left) and an equivalent normalized one (right).

$$\{D_1, (\mathsf{OR}, (\mathsf{AND}, (\{K_5^1\}_{K_1}, \{K_5^2\}_{K_1}, \{K_6\}_{K_5}), \{K_6\}_{K_4}, \{D_2, (\{D_3\}_{K_3}, \{D_4\}_{K_4})\}_{K_6}), D_5, (\{D_6\}_{K_2}))\}_{K_1}$$

Figure 2: Expression associated with the normalized protection of Figure 1.

the set Keys or key shares in the set KeyShares defined by:

$$\mathsf{KeyShares} = \{K_i^j \mid K_i \in \mathsf{Keys}, j \in 1..n\}$$

for a fixed parameter n. Thus, in summary, protections are generated by the grammar:

$$\begin{array}{rcl} \mathsf{Prot} & ::= & [(\mathsf{BData}), \mathsf{Cond}] \mid \\ & & [(\mathsf{BData}, \mathsf{Prot}, \mathsf{Prot}, \ldots, \mathsf{Prot}), \mathsf{Cond}] \\ \mathsf{Cond} & ::= & \mathsf{true} \mid \mathsf{false} \mid \mathsf{Keys} \mid \mathsf{Cond} \wedge \mathsf{Cond} \mid \mathsf{Cond} \vee \mathsf{Cond} \end{array}$$

where

$$\mathsf{BData} = \mathsf{Data} \cup \mathsf{Keys} \cup \mathsf{KeyShares} \cup \{\mathsf{OR}, \mathsf{AND}\}$$

is the set of node labels.

Roughly, the key shares K_i^1, K_i^2, $\ldots$, K_i^n are pieces of information that together allow the recovery of the key K_i, but of which no proper subset suffices for computing K_i, or even non-trivial partial information about K_i. For example, when $n = 2$, the key share K_i^1 may be a random string of the same length as K_i, and K_i^2 may be the XOR of K_i with K_i^1. We treat a framework more general than the original one, which uses a particular way of sharing keys. We assume that the number of shares for each key is some constant n, and that each key is shared only once. (This assumption holds in the original framework, where $n = 2$.)

In a protection, the formula that guards a node describes a condition that a user needs to satisfy for accessing that node (and its children). For example, accessing a node guarded by the formula $K_1 \wedge (K_2 \vee K_3)$ requires having key K_1 and at least one of the keys K_2 and K_3. In addition, the user should also satisfy all formulas on the path from the root to the node.

Using simple transformations, one can rewrite any protection into an equivalent, *normalized* protection where all formulas that guard nodes are atomic, that is, one of true, false, or K for some $K \in \mathsf{Keys}$. Normalization requires adding metadata nodes, keys, and key shares. Normalization can also include removing parts guarded by false, so we assume that false does not appear in normalized formulas (departing slightly from the original definition but without loss of generality). Normalized protections are important because, in standard encryption schemes, one can encrypt under an atomic key but not under a boolean combination of keys. Normalized protections serve as the basis for producing partially encrypted documents by applying an encryption algorithm repeatedly.

We explain the semantics of protections and the role of key sharing using the example in Figure 1 taken from [19]. Notice that we preserved the original labels—we did not replace element names with symbols D_1, D_2, $\ldots$. The tree on the left is an example of a protection over some XML medical database. A user that possesses only key K_3 cannot access any node since the root is guarded by K_1. If the user knows also K_1 then it should be able to access the data in nodes $\{1, 2, 3, 5\}$. The tree on the right is a normalized protection, equivalent to the protection on the left. In the normalized protection, a user with keys K_1 and K_3 can recover the information in nodes 9 and 10, that is, the key shares K_5^1 and K_5^2, and therefore K_5. (Recall that, in [19], keys are split into only two shares.) The key K_5 can then be used to recover the information in node 11, that is, the key K_6 which together with K_1 provides access to the nodes $\{1, 2, 3, 5\}$ of the original tree.

Formally, the semantics of a protection P is the function $\mathsf{Acc}_P : \mathcal{P}(\mathsf{Keys}) \rightarrow \mathcal{P}(\mathsf{Data})$ that, given a set of keys $T \subseteq \mathsf{Keys}$, returns the set of data that can be accessed using the keys in T. Computing the function $\mathsf{Acc}_P(T)$ is an iterative process. The keys in T are used to access new keys (by either obtaining them directly or recovering all their shares), and the process is repeated until a maximal set of keys is obtained. The output of $\mathsf{Acc}_P(T)$ is the set of all data contained in nodes that can be accessed using this last set of keys.

In the description above, and in the analysis that follows, we do not consider the use of keys derived from data (for example, from mother's maiden names and social security numbers). There are at least three obstacles to obtaining security guarantees with such keys. These obstacles are not specific to a particular scheme, but they do arise in this context. First, the potential lack of entropy in data implies that the resulting keys may offer no security. Moreover, the key derivations can lose some data entropy. Finally, the resulting keys can be involved in questionable encryption cycles—for instance, protecting a mother's maiden name with a social security number and vice versa.

3. FORMAL ANALYSIS

In this section we introduce a language for representing cryptographic expressions that use symmetric encryption and secret sharing schemes. We rely on this language for providing a formal account of the transformation of protections into partially encrypted documents. We also define patterns of expressions, which capture the information that expressions reveal to an adversary, in symbolic terms. (This definition is a variant of ones suggested in other contexts [1, 13, 17].) Finally, we establish a relation between the semantics of protections and these patterns.

An intermediate cryptographic language

We consider expressions built from the set of basic data BData (defined above), using tupling and encryption with keys in Keys. The grammar for expressions is:

$$\mathsf{Exp} ::= \mathsf{BData} \mid (\mathsf{Exp}, \mathsf{Exp}, \ldots, \mathsf{Exp}) \mid \{\mathsf{Exp}\}_{\mathsf{Keys}}$$

For example, an element in Exp is the expression

$$(\{(D_2, K_1^2)\}_{K_2}, \{(D_1, \{K_2\}_{K_3})\}_{K_1})$$

It represents the tupling of two ciphertexts. The first ciphertext is the encryption under key K_2 of data D_2 and key share K_1^2. The second ciphertext is the encryption under key K_1 of the tuple formed from data D_1 and the encryption under key K_3 of key K_2.

Associating cryptographic expressions to protections

We use our language of expressions for giving a precise definition of how to map a normalized protection to (a symbolic representation for) a partially encrypted document. This definition is recursive. It relies on the intuition that each node of the XML tree under consideration is (recursively) mapped to a part of the final, partially encrypted document. If the node is guarded by a key, then the corresponding part of the final document is encrypted under that key, otherwise it is left in clear.

Formally, if P_i (for $i = 1, 2, \ldots, l$) are normalized protections, B is an arbitrary basic data symbol, and K is an arbitrary key in Keys, we define the function $\mathsf{E} : \mathsf{Prot} \to \mathsf{Exp}$ by:

- $\mathsf{E}([(B), \mathsf{true}]) = (B)$
- $\mathsf{E}([(B, P_1, P_2, \ldots, P_l), \mathsf{true}]) =$ $(B, \mathsf{E}(P_1), \mathsf{E}(P_2), \ldots, \mathsf{E}(P_l))$
- $\mathsf{E}([(B), K]) = \{(B)\}_K$
- $\mathsf{E}([(B, P_1, P_2, \ldots, P_l), K]) =$ $\{(B, \mathsf{E}(P_1), \mathsf{E}(P_2), \ldots, \mathsf{E}(P_l))\}_K$

For example, the expression of Figure 2 corresponds to the second protection in Figure 1, with the element name contained in node i replaced with data symbol D_i.

Cycles

The definition of expressions allows encryption cycles of the kind discussed in the introduction. For some of our results, it is useful to focus on a class of acyclic expressions:

Definition 1 (Acyclic expressions). *A plain occurrence of a key K_j in an expression E_1 is one where K_j is not the subscript in a subexpression $\{\ldots\}_{K_j}$. Key K_i encrypts key K_j in expression E if there exists a subexpression $\{E_1\}_{K_i}$ in E such that K_j occurs plainly in E_1 or K_j^k (for some $k \in 1..n$) occurs in E_1. The expression E is acyclic if the graph associated with its "encrypts" relation is acyclic.*

For example, $\{K_1\}_{K_2}$ and $\{\{K_1\}_{K_2}\}_{K_2}$ are both acyclic, while $\{K_1\}_{K_1}$ and $(\{K_1^1\}_{K_2}, \{K_2\}_{K_1})$ are not.

As long as data values are not used as keys, a protection P that results from the translation of a policy never contains keys in its nodes. In turn, if normalizing P yields P', then $\mathsf{E}(P')$ is an acyclic expression. (This point follows from the definition of normalization [19].)

Recoverable keys

Given an expression E, we write $\mathsf{keys}(E)$ for the set of key symbols that occur in E or key symbols whose shares occur in E. We call a key *recoverable* if the key occurs in clear (that is, not encrypted), if the key occurs encrypted under recoverable keys, or if all its shares occur in clear or encrypted under recoverable keys. Thus, the set of recoverable keys is defined inductively. We write $\mathsf{recoverable}(E)$ for the set of all recoverable keys in expression E. For example, with $n = 2$, for the expression

$$E = (\{K_1, K_2^1, K_6^1, K_4\}_{K_3}, \{K_2^2\}_{K_2}, K_3, \{K_5\}_{K_4})$$

we have

$$\mathsf{keys}(E) = \{K_1, K_2, K_3, K_4, K_5, K_6\}$$

and

$$\mathsf{recoverable}(E) = \{K_1, K_3, K_4, K_5\}$$

while for the expression

$$E' = (\{K_1, K_2^1, K_6^1, K_4\}_{K_3}, \{K_5\}_{K_2}, K_3, \{K_2^2\}_{K_4})$$

we have

$$\mathsf{keys}(E') = \{K_1, K_2, K_3, K_4, K_5, K_6\}$$

and

$$\mathsf{recoverable}(E') = \{K_1, K_2, K_3, K_4, K_5\}$$

Expression patterns

For each expression E, we define its *structure* $\mathsf{struct}(E)$. Essentially, the structure of an expression is given by its parse tree in which the labels are replaced with fresh symbols. We therefore introduce D, K_0, and K_0^j (for $j \in 1..n$), all disjoint from BData, and define the structure of expressions by:

- $\mathsf{struct}(K_i) = K_0$ for all $K_i \in \mathsf{Keys}$;
- $\mathsf{struct}(K_i^j) = K_0^j$ for all $K_i \in \mathsf{Keys}$, $j \in 1..n$;
- $\mathsf{struct}(D_i) = D$ for all $D_i \in \mathsf{Data}$;
- $\mathsf{struct}(\mathsf{OR}) = \mathsf{OR}$;
- $\mathsf{struct}(\mathsf{AND}) = \mathsf{AND}$;

- $\mathsf{struct}((E_1, E_2, \ldots, E_m)) =$
  $(\mathsf{struct}(E_1), \mathsf{struct}(E_2), \ldots, \mathsf{struct}(E_m))$;
- $\mathsf{struct}(\{E\}_{K_i}) = \{\mathsf{struct}(E)\}_{K_0}$ for all $K_i \in \mathsf{Keys}$.

Thus, data symbols are replaced with D, key symbols are replaced with K_0, and key shares are replaced with corresponding shares of K_0. For example, the structure of the expression

$$\{(\{K_1, D_2\}_{K_3}, K_3^1)\}_{K_2}$$

is

$$\{(\{K_0, D\}_{K_0}, K_0^1)\}_{K_0}$$

The encryption cycle in this structure is unimportant for our purposes. Alternative definitions of expression structure can avoid such cycles without affecting our results. (For example, one such definition lets the structure of $\{E\}_{K_i}$ be $\{\mathsf{struct}(E)\}_{K_0'}$, where K_0' is a fresh symbol distinct from K_0.)

We write $\mathsf{p}(E, T)$ for the *pattern* that can be observed in expression E using for decryption the keys in $T \subseteq \mathsf{keys}(E)$. The set of patterns is defined like the set of expressions but over extended sets of atomic symbols that include D, K_0, and K_0^j (for $j \in 1..n$). The pattern $\mathsf{p}(E, T)$ is defined by:

- $\mathsf{p}(B, T) = B$ for all $B \in \mathsf{BData}$;
- $\mathsf{p}((E_1, E_2, \ldots, E_m), T) =$
  $(\mathsf{p}(E_1, T), \mathsf{p}(E_2, T), \ldots, \mathsf{p}(E_m, T))$;
- $\mathsf{p}(\{E\}_{K_i}, T) = \{\mathsf{struct}(E)\}_{K_i}$ if $K_i \notin T$;
- $\mathsf{p}(\{E\}_{K_i}, T) = \{\mathsf{p}(E, T)\}_{K_i}$ if $K_i \in T$.

The idea that motivates this definition is that ciphertexts encrypted under unknown keys reveal at most the structure of the underlying plaintext, but not the encrypted values.

We write $\mathsf{pattern}(E)$ for the pattern obtained from E by using for decryption the keys recoverable from E itself. We let:

$$\mathsf{pattern}(E) = \mathsf{p}(E, \mathsf{recoverable}(E))$$

For example, the pattern of the expression

$$(\{D_1\}_{K_1}, \{D_2\}_{K_2}, K_1)$$

is

$$(\{D_1\}_{K_1}, \{D\}_{K_2}, K_1)$$

Similarly, the pattern of the expression

$$(\{\{D_1\}_{K_2}\}_{K_1}, \{D_2\}_{K_2}, K_1)$$

is

$$(\{\{D\}_{K_2}\}_{K_1}, \{D\}_{K_2}, K_1)$$

Finally, when $n = 2$, the pattern of the expression

$$(\{D_1\}_{K_1}, \{\{K_1\}_{K_2}\}_{K_3}, K_3^1, K_3^2, \{(\{K_1, D_2\}_{K_3}, K_3^1)\}_{K_2})$$

is

$$(\{D\}_{K_1}, \{\{K_0\}_{K_2}\}_{K_3}, K_3^1, K_3^2, \{(\{K_0, D\}_{K_0}, K_0^1)\}_{K_2})$$

The abstract semantics of a protection P and the pattern of the document $\mathsf{E}(P)$ derived from P are related by the following theorem. This theorem states that if, according to protection P, data D_i can be rightfully accessed using a set of keys $\{K_1, K_2, \ldots, K_l\}$, then D_i occurs in the pattern observed in $(\mathsf{E}(P), K_1, K_2, \ldots, K_l)$; and, conversely, if D_i occurs in the pattern observed in $(\mathsf{E}(P), K_1, K_2, \ldots, K_l)$, then D_i can be rightfully accessed using that set of keys according to P.

THEOREM 1. *Let P be a normalized protection. For any set of keys $T = \{K_1, K_2, \ldots, K_l\} \subseteq \mathsf{keys}(\mathsf{E}(P))$ and any $D_i \in \mathsf{Data}$, it holds that $D_i \in \mathsf{Acc}_P(T)$ if and only if D_i occurs in $\mathsf{pattern}((\mathsf{E}(P), K_1, K_2, \ldots, K_l))$.*

Again, alternative sets of definitions are certainly possible, and perhaps attractive. Those definitions may capture stronger security properties. With them, we may require that a ciphertext encrypted under an unknown key reveal nothing about the underlying plaintext (not even its structure or its length) [1]. For instance, we may let:

- $\mathsf{p}'(B, T) = B$ for all $B \in \mathsf{BData}$;
- $\mathsf{p}'((E_1, E_2, \ldots, E_m), T) =$
  $(\mathsf{p}'(E_1, T), \mathsf{p}(E_2, T), \ldots, \mathsf{p}'(E_m, T))$;
- $\mathsf{p}'(\{E\}_{K_i}, T) = \{\square\}_{K_i}$ if $K_i \notin T$;
- $\mathsf{p}'(\{E\}_{K_i}, T) = \{\mathsf{p}'(E, T)\}_{K_i}$ if $K_i \in T$.

where $\square$ is a special symbol that represents undecryptable material, and

$$\mathsf{pattern}'(E) = \mathsf{p}'(E, \mathsf{recoverable}(E))$$

In another variant, we may let $\mathsf{p}'(\{E\}_{K_i}, T) = \square$ if $K_i \notin T$, hiding the occurrence of the key K_i in the resulting pattern.

Theorem 1 holds with these and other variants of the definitions. However, the other theorems of this paper are more problematic in this respect. We return to this point in Section 4.

4. COMPUTATIONAL ANALYSIS

In this section we give a concrete interpretation for expressions and patterns as probability distributions on bit-strings. This interpretation is computational in the sense that it relies on computations on bit-strings rather than on symbolic expressions. In particular, encryption is an actual computation on bit-strings, rather than a formal operation. In this computational world, we give a cryptographic definition for data secrecy, then prove our main results.

We first recall the definition of computational indistinguishability (a central ingredient for defining cryptographic secrecy) and those of encryption and secret sharing schemes.

Indistinguishability of distribution ensembles

Let $\{D_\eta^0\}_\eta$ and $\{D_\eta^1\}_\eta$ be two distribution ensembles (sequences of probability distributions, indexed by a security parameter η) and A an algorithm. We write $x \overset{R}{\leftarrow} D_\eta^i$ to indicate that x is sampled according to D_η^i. The advantage of A in distinguishing between the two ensembles is the quantity:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{dist}}_{D_0,D_1}(A,\eta) \quad &= \quad \Pr\left[x \overset{R}{\leftarrow} D_\eta^0 : A(x,\eta) = 1\right] - \\ &\qquad \Pr\left[x \overset{R}{\leftarrow} D_\eta^1 : A(x,\eta) = 1\right]\end{aligned}$$

We say that D_0 and D_1 are computationally indistinguishable, and we write $D_0 \approx D_1$, if for any probabilistic polynomial-time algorithm A the quantity $\mathbf{Adv}^{\mathrm{dist}}_{D_0,D_1}(A,\eta)$ is negligible as a function of η. (A function $f(\eta)$ is negligible if it is smaller than the inverse of any polynomial for all sufficiently large inputs η [3].)

Encryption schemes

An encryption scheme Π consists of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ for key generation, encryption, and decryption, respectively. The key generation algorithm is randomized; it takes as input a security parameter η and returns a key k to be used for both encryption and decryption. We write $k \overset{R}{\leftarrow} \mathcal{K}(\eta)$ for the process of generating encryption keys. The encryption algorithm is also randomized. It takes as input a key k and a plaintext m, and outputs a ciphertext c. We write $c \overset{R}{\leftarrow} \mathcal{E}(k, m)$ for the process of encrypting message m with key k, producing c. Finally, the decryption algorithm takes as input a key k and a ciphertext c. If $c \overset{R}{\leftarrow} \mathcal{E}(k, m)$ for a key k and a plaintext m, then $\mathcal{D}(k, c) = m$. Decryption returns $\perp$ if it does not succeed.

We use a standard notion of security for encryption: indistinguishability against chosen plaintext attacks ($\mathsf{IND\text{-}CPA}$), a.k.a. semantic security [12]. We define it next. For a given encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and a bit b, we consider a "left-right" oracle $LR_{\Pi,b}(\eta)$. This oracle is a program that generates a key k (via $k \overset{R}{\leftarrow} \mathcal{K}(\eta)$), and then answers queries of the form (m_0, m_1), where m_0 and m_1 are bit-strings of equal length. The oracle always returns the answer $\mathcal{E}(k, m_b)$, that is, the encryption of the message selected by b. Scheme Π is said to be $\mathsf{IND\text{-}CPA}$ secure if for any probabilistic polynomial-time adversary A with access to the oracle described above, the quantity:

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\mathrm{ind-cpa}}(A, \eta) \quad = \quad & \Pr[A^{LR_{\Pi,0}}(\eta) = 1] - \\ & \Pr[A^{LR_{\Pi,1}}(\eta) = 1]\end{aligned}$$

is negligible as a function of η. Intuitively, the adversary A does not know b a priori, and it aims to discover b. The advantage $\mathbf{Adv}_{\Pi}^{\mathrm{ind-cpa}}(A, \eta)$ will not be negligible if A can determine b by interacting with the oracle. A fortiori, the advantage $\mathbf{Adv}_{\Pi}^{\mathrm{ind-cpa}}(A, \eta)$ will not be negligible if A can break encryptions, and therefore determine b by decrypting the oracle's response to a query (m_0, m_1) where $m_0 \neq m_1$.

According to this definition of security, encryption need not hide the length of plaintexts (because m_0 and m_1 are required to be of equal length). Furthermore, encryption need not hide the identity of keys: an adversary may be able to distinguish two encryptions under the same unknown key from two encryptions under different unknown keys. Alternative notions of security can yield stronger guarantees, but correspondingly they are harder to satisfy.

Secret sharing schemes

An n-out-of-n secret sharing scheme $\mathcal{SS} = (\mathcal{S}, \mathcal{C})$ for sharing keys of Π consists of algorithms for share creation and share combination. The randomized share creation algorithm $\mathcal{S}$ takes as input a key k and the security parameter η used to generate it, and outputs n shares of k: k^1, k^2, ..., k^n. The share combination algorithm $\mathcal{C}$ takes as input n shares $k^1, k^2, \ldots, k^n$, and attempts to reconstitute the original key. The scheme is correct if $\mathcal{C}(\mathcal{S}(k, \eta)) = k$ for any key k and any η.

Security of secret sharing requires that proper subsets of shares for any two keys are indistinguishable. Let $sh(k)$ be a sample from the distribution $\mathcal{S}(k, \eta)$ of the n shares output by the sharing algorithm for the key k, and $sh(k)|_S$ be the restriction of $sh(k)$ to the indexes in some set of indexes $S \subseteq \{1, 2, \ldots, n\}$. We require that for any pair of keys k_0 and k_1 output by $\mathcal{K}(\eta)$, for any proper subset of indexes $S \subset \{1, 2, \ldots, n\}$, and for any probabilistic polynomial-time adversary A, the quantity:

$$\begin{aligned}&\mathbf{Adv}_{\mathcal{SS}}^{\mathrm{ss}}(A, \eta) = \\ &\qquad \Pr[sh(k_0) \overset{R}{\leftarrow} \mathcal{S}(k_0, \eta) : A(sh(k_0)|_S) = 1)] - \\ &\qquad \Pr[sh(k_1) \overset{R}{\leftarrow} \mathcal{S}(k_1, \eta) : A(sh(k_1)|_S) = 1)]\end{aligned}$$

is negligible as a function of η.

Computational interpretation of expressions

Expressions and patterns induce distributions on bit-strings. These distributions are obtained by replacing data symbols with bit-strings and implementing encryption and key sharing with actual encryption and secret sharing schemes.

Formally, for any expression or pattern E, given a function $f : \mathsf{Data} \rightarrow \{0,1\}^*$ that maps data symbols to bit-string representations of XML element names and values, an encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, a secret sharing key $\mathcal{SS} = (\mathcal{S}, \mathcal{C})$, and a security parameter η, we define the distribution $[\![E]\!]_f^{\Pi,\mathcal{SS},\eta}$ (and thus a distribution ensemble $[\![E]\!]_f^{\Pi,\mathcal{SS}}$) using a two-step procedure:

1. In the first step, we map each key symbol to a bit-string. Specifically, we assume that $\mathsf{keys}(E) = \{K_1, \ldots, K_m\}$, and we generate a vector τ from the distribution $\mathcal{K}^{m+1}(\eta)$. This is a vector of $m + 1$ keys, each obtained by running the key generation algorithm on the security parameter. We map K_i to $\tau[i]$, and in particular map K_0 to $\tau[0]$.

   We then obtain shares of the keys in τ by running the secret sharing scheme $\mathcal{SS}$. These shares are maintained in an $(m + 1)$-by-n matrix ϕ whose rows are obtained by $\phi[i] \overset{R}{\leftarrow} \mathcal{S}(\tau[i])$.

2. In the second step, we map each expression (or pattern) E to an interpretation $[\![E]\!]_f^{\Pi,\mathcal{SS},\eta}$ that we define inductively in Figure 3. This step assumes constant bit-strings "`or`", "`and`", and "`data`", as well as a tupling operation on bit-strings.

The computational interpretation $[\![\mathsf{E}(P)]\!]_f^{\Pi,\mathcal{SS},\eta}$ associated with the expression $\mathsf{E}(P)$ is the distribution of the partially encrypted document derived from protection P, generated with encryption scheme Π, secret sharing scheme $\mathcal{SS}$, and η as security parameter.

Because encryption may not hide the length of plaintexts, we typically need hypotheses on the lengths of bit-string representations. For simplicity, we assume that those lengths depend only on structure. More precisely, we assume that the lengths of all samples from the distributions $[\![E]\!]_f^{\Pi,\mathcal{SS},\eta}$ and $[\![\mathsf{struct}(E)]\!]_f^{\Pi,\mathcal{SS},\eta}$ are equal.

In the case where E is a data symbol, this assumption means that we focus on functions $f : \mathsf{Data} \rightarrow \{0,1\}^*$ that map data symbols and "`data`" to bit-strings of a fixed length:

DEFINITION 2. *A valuation is a function $f : \mathsf{Data} \rightarrow \{0,1\}^*$ that maps every data symbol to a bit-string of the same length as* `data`.

In the case where E is a key symbol, the assumption means that the key generator yields keys of a fixed length for each given security parameter. A similar condition applies to key shares. In the case where E is a tuple, it suffices that

$$
\begin{aligned}
\llbracket D \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= \texttt{data} \\
\llbracket D_i \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= f(D_i) \\
\llbracket \mathsf{OR} \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= \texttt{or} \\
\llbracket \mathsf{AND} \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= \texttt{and} \\
\llbracket K_i \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= \tau[i] \\
\llbracket K_i^j \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= \phi[i][j] \\
\llbracket (E_1, E_2, \dots, E_l) \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= (\llbracket E_1 \rrbracket_f^{\Pi,\mathcal{SS},\eta}, \llbracket E_2 \rrbracket_f^{\Pi,\mathcal{SS},\eta}, \dots, \llbracket E_l \rrbracket_f^{\Pi,\mathcal{SS},\eta}) \\
\llbracket \{E\}_{K_i} \rrbracket_f^{\Pi,\mathcal{SS},\eta} &= \mathcal{E}(\tau[i], \llbracket E \rrbracket_f^{\Pi,\mathcal{SS},\eta})
\end{aligned}
$$

Figure 3: Mapping expressions to bit-strings.

the length of a tuple is a function of the lengths of its components. Similarly, in the case where E is an encryption, it suffices that the length of a ciphertext is a function of the length of the underlying plaintext and of the security parameter. These conditions on keys, key shares, tuples, and encryptions hold in most usual implementations.

Secrecy, computationally

We use the computational interpretation of expressions for giving a computational characterization of the secrecy of data that occurs in expressions:

Definition 3. *Let E be an expression, Π an encryption scheme, $\mathcal{SS}$ a secret sharing scheme, and $S \subseteq \mathsf{Data}$. The set S is computationally hidden in E (with Π and $\mathcal{SS}$) if for any two valuations f_0 and f_1 such that*

$$f_0(D_i) = f_1(D_i) \quad \textit{for all } D_i \in \mathsf{Data} - S$$

it holds that $\llbracket E \rrbracket_{f_0}^{\Pi,\mathcal{SS}} \approx \llbracket E \rrbracket_{f_1}^{\Pi,\mathcal{SS}}$.

As explained in the introduction, this definition supposes that an adversary is allowed to choose two interpretations f_0 and f_1 for the data symbols in the expression E. These interpretations must map data symbols that are not secret to the same bit-strings, but may map other data symbols to different bit-strings of the same length. The adversary is then given a bit-string selected from the distribution induced by one of the two interpretations, and its goal is to determine which interpretation was used. Secrecy means that the adversary cannot do much better than guessing.

Main results

The technical core of our results is the next theorem. It states that patterns faithfully represent the information that expressions reveal, even when expressions and patterns are mapped to bit-strings. Specifically, we prove that the distribution ensembles associated with E and $\mathsf{pattern}(E)$ are indistinguishable.

Theorem 2. *Let E be an acyclic expression. If Π is an* IND-CPA *secure encryption scheme and $\mathcal{SS}$ is a secure secret sharing scheme, then for any valuation f it holds that*

$$\llbracket E \rrbracket_f^{\Pi,\mathcal{SS}} \approx \llbracket \mathsf{pattern}(E) \rrbracket_f^{\Pi,\mathcal{SS}}$$

The proof of this theorem relies on a so-called hybrid argument. It is presented in the Appendix.

Building on this theorem, Theorem 3 relates the abstract semantics of a normalized protection P, as defined by the function $\mathsf{Acc}_P(\cdot)$, to the secrecy of data in the partially encrypted document associated with P. It requires that $\mathsf{E}(P)$ be acyclic, as we would expect for protections derived from policies (see Section 3). It states that if some data is secret according to the abstract semantics of protections, then that data is in fact computationally hidden. Therefore, we regard Theorem 3 as the main theorem of this paper.

Theorem 3. *Let P be a normalized protection such that $\mathsf{E}(P)$ is an acyclic expression. Let $T = \{K_1, K_2, \dots, K_l\} \subseteq \mathsf{keys}(\mathsf{E}(P))$ be an arbitrary set of keys. If Π is an* IND-CPA *secure encryption and $\mathcal{SS}$ is a secure secret sharing scheme, then $\mathsf{Data}-\mathsf{Acc}_P(T)$ is computationally hidden in $(\mathsf{E}(P), K_1, K_2, \dots, K_l)$ with Π and $\mathcal{SS}$.*

This theorem follows from Theorem 1 and Theorem 2. If $D_i \notin \mathsf{Acc}_P(T)$, Theorem 1 implies that D_i does not occur in $\mathsf{pattern}(\mathsf{E}(P), K_1, K_2, \dots, K_l)$, so:

$$\llbracket \mathsf{pattern}((\mathsf{E}(P), K_1, K_2, \dots, K_l)) \rrbracket_{f_0}^{\Pi,\mathcal{SS}} = \llbracket \mathsf{pattern}((\mathsf{E}(P), K_1, K_2, \dots, K_l)) \rrbracket_{f_1}^{\Pi,\mathcal{SS}}$$

for any valuations f_0 and f_1 that coincide on $\mathsf{Acc}_P(T)$. We conclude by Theorem 2 and transitivity.

Theorems 2 and 3 allow for the possibility that an attacker may be able to learn a great deal about the length and even the structure of encrypted material. Under the standard definition of security that we adopt for encryption, one cannot expect to do much better [16]. As explained above, encryption need not hide the length of plaintexts. Moreover, if E and E' are expressions with different structures, the concrete bit-string implementations $\llbracket E \rrbracket_f^{\Pi,\mathcal{SS},\eta}$ and $\llbracket E' \rrbracket_f^{\Pi,\mathcal{SS},\eta}$ may well have different lengths, so it is possible that their encryptions could be distinguished.

In particular, analogues of Theorem 2 would not hold for the alternative functions $\mathsf{pattern}'$ defined in Section 3. Stronger assumptions on encryption would yield those analogues, and corresponding strengthenings of Theorem 3.

5. CONCLUSION

The main contribution of this paper is a precise justification of the encryption-based techniques for enforcing access policies for XML documents, as developed by Miklau and Suciu. More specifically, we provide a proof that XML data that is secret according to an abstract, symbolic semantics is indeed secret with respect to a strong, computational notion of security.

In defining the subject of our analysis, we have attempted to be faithful to the work of Miklau and Suciu. In further research, one might wish to depart from their framework. In particular, like them, we have focused on protection against off-line attacks. In such attacks, the adversary obtains information about a document only by observing it. In further work, it may be interesting to consider active attacks, in which the adversary may interactively influence the document structure and contents. It may also be interesting to consider richer access control policies, as well as richer data models with key constraints, functional dependencies, and other refinements. The value of rigorous analysis may be even larger with these enrichments, but our basic approach should remain applicable and helpful in bridging the gap between high-level designs and precise guarantees.

Acknowledgments

We are grateful to Véronique Cortier, Phokion Kolaitis, Daniele Micciancio, Gerome Miklau, Dan Suciu, and Victor Vianu for helpful discussions on this work and its presentation.

This work was supported in part by the National Science Foundation under Grants CCR-0204162, CCR-0208800, and ITR-0430594.

6. REFERENCES

[1] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[2] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *Proc. of the 10th ACM Conference on Computer and Communications Security*, pages 220–330. ACM Press, 2003. Long version: IACR ePrint Archive, Report 2003/015.

[3] Mihir Bellare and Phil Rogaway. Introduction to modern cryptography. Available at: `http://www.cs.ucsd.edu/~mihir/cse207/classnotes.html`.

[4] Elisa Bertino, B. Carminati, and E. Ferrari. A temporal key management scheme for secure broadcasting of XML documents. In *Proc. of the 8th ACM Conference on Computer and Communication Security*, pages 31–40, 2002.

[5] Elisa Bertino, Silvana Castano, and Elena Ferrari. Author-X: A comprehensive system for securing XML documents. *IEEE Internet Computing*, 5(3):21–31, 2001.

[6] Silvana Castano, Mariagrazia G. Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. Addison-Wesley – ACM Press, 1995.

[7] Jason Crampton. Applying hierarchical and role-based access control to XML documents. In *Proc. of ACM Workshop on Secure Web Services*, pages 41–50, 2004.

[8] Ernesto Damiani, Sabrina de Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 5(2):169–202, 2002.

[9] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. *SIAM Journal of Computing*, 30(2):391–437, 2000.

[10] Donald Eastlake and Joseph Reagle. XML encryption syntax and processing. `http://www.w3.org/TR/xmlenc-core`, October 2002.

[11] David K. Gifford. Cryptographic sealing for information secrecy and authentication. *Commun. ACM*, 25(4):274–286, 1982.

[12] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, April 1984.

[13] Jonathan Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, Massachusetts Institute of Technology, 2004.

[14] Michiharu Kudo and Satoshi Hada. XML document security based on provisional authorization. In *Proc. the 7th ACM Conference on Computer and Communication Security*, pages 87–96, 2000.

[15] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. of 2004 IEEE Symposium on Security and Privacy*, pages 71–85, 2004.

[16] Daniele Micciancio. Towards computationally sound symbolic security analysis. Talk at DIMACS; slides available at: `http://dimacs.rutgers.edu/Workshops/Protocols/slides/micciancio.pdf`, 2004.

[17] Daniele Micciancio and Saurabh Panjwani. Adaptive security of symbolic encryption. In Joe Kilian, editor, *Theory of cryptography conference - Proceedings of TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 169–187. Springer-Verlag, February 2005.

[18] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In Moni Naor, editor, *Theory of cryptography conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, February 2004.

[19] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB 2003: Proc. of 29th International Conference on Very Large Data Bases*, pages 898–909, 2003.

[20] John Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols. *Electronic Notes in Theoretical Computer Science*, 45, 2001.

[21] Adi Shamir. How to share a secret. *CACM*, 22(11):612–613, 1979.

[22] Jeffrey Ullman. *Principles of Database Systems*. Computer Science Press, Potomac, MD, 1983.

[23] Xiaochung Yang and Chen Li. Secure XML publishing without information leakage in the presence of data inference. In *VLDB 2004: Proc. of 30th International Conference on Very Large Data Bases*, pages 96–107, 2004.

APPENDIX

Proof of Theorem 2

In this appendix we provide a proof of Theorem 2. Some aspects of the proof are by now standard; the reader may wish to consult the proof of the main theorem in the work of Abadi and Rogaway [1]. We start with some notation and a couple of useful lemmas.

A permutation $\sigma : \mathsf{Keys} \cup \mathsf{KeyShares} \rightarrow \mathsf{Keys} \cup \mathsf{KeyShares}$ is *consistent* if

- $\sigma(K_i) \in \mathsf{Keys}$ for all $K_i \in \mathsf{Keys}$, and
- if $K_i = \sigma(K_j)$, then $K_i^l = \sigma(K_j^l)$ for all $l \in \{1, 2, \ldots, n\}$.

For any consistent permutation σ, we write $E\sigma$ for the expression obtained from E by renaming its keys and key shares according to σ. Consistent renamings do not change the distribution ensembles associated with expressions:

LEMMA 4. *For any expression E, if σ is a consistent permutation then $\llbracket E \rrbracket_f^{\Pi,\mathcal{SS}} \approx \llbracket E\sigma \rrbracket_f^{\Pi,\mathcal{SS}}$.*

This lemma holds because the output of the algorithm that associates a distribution with an expression does not depend on the actual atomic symbols used in the expression, but only on their meaning.

For each expression and pattern E, we write $\mathsf{hidden}(E)$ for the set of keys that are not recoverable from E:

$$\mathsf{hidden}(E) = \mathsf{keys}(E) - \mathsf{recoverable}(E)$$

The following lemma states that the keys in any acyclic expression E can be reordered so that the first l keys are the keys hidden in E and the remaining keys are the keys that can be recovered from E. Moreover, it is possible to reorder the keys so that, in the resulting expression, for any two hidden keys K_i and K_j, if K_i encrypts K_j then $i < j$.

LEMMA 5. *If E is an acyclic expression, $m = |\mathsf{keys}(E)|$, and $l = |\mathsf{hidden}(E)|$, then there exists a consistent permutation σ such that:*

- $\mathsf{keys}(E\sigma) = \{K_1, K_2, \ldots, K_l, K_{l+1}, \ldots, K_m\}$,
- $\mathsf{hidden}(E\sigma) = \{K_1, K_2, \ldots, K_l\}$,
- *for any $1 \leq i, j \leq l$, if K_i encrypts K_j in $E\sigma$ then $i < j$.*

By Lemmas 4 and 5, it is sufficient to prove the theorem under the following additional assumptions:

- $\mathsf{keys}(E) = \{K_1, K_2, \ldots, K_l, K_{l+1}, \ldots, K_m\}$,
- $\mathsf{hidden}(E) = \{K_1, K_2, \ldots, K_l\}$,
- for any $1 \leq i, j \leq l$, if K_i encrypts K_j then $i < j$.

Proof of the theorem

We prove the theorem by a hybrid argument: given an expression E as above, we exhibit a series of distribution ensembles $D_0, D_1, \ldots, D_l$, such that:

1. $D_0 = \llbracket E \rrbracket_f^{\Pi,\mathcal{SS}}$,
2. $D_l = \llbracket \mathsf{pattern}(E) \rrbracket_f^{\Pi,\mathcal{SS}}$, and
3. $D_{i-1} \approx D_i$, for all $1 \leq i \leq l$.

Since l is a fixed constant (independent of the security parameter), the conclusion of the theorem immediately follows.

Consider the sequence of patterns E_0, E_1, $\ldots$, E_l inductively defined by:

- $E_0 = E$,
- E_i is obtained by replacing each subexpression of E_{i-1} that is of the form $\{E\}_{K_i}$ with $\{\mathsf{struct}(E)\}_{K_i}$.

For each $0 \leq i \leq l$ we let D_i be the distribution ensemble associated with pattern E_i and prove that the resulting sequence of distribution ensembles satisfies conditions 1–3. It is immediate that $D_0 = \llbracket E \rrbracket_f^{\Pi,\mathcal{SS}}$. Also, since pattern E_l is obtained by replacing all subexpressions of E of the form $\{E\}_{K_i}$ for some $K_i \in \mathsf{hidden}(E)$ with $\{\mathsf{struct}(E)\}_{K_i}$, we have that $E_l = \mathsf{pattern}(E)$ and, therefore, that $D_l = \llbracket \mathsf{pattern}(E) \rrbracket_f^{\Pi,\mathcal{SS}}$. It only remains to be shown that for each $1 \leq i \leq l$, the distribution ensembles D_{i-1} and D_i are indistinguishable (condition 3). For each $0 \leq i \leq l-1$ we introduce two intermediate distribution ensembles $D_{i,0}$ and $D_{i,1}$ and we prove that:

$$D_i \approx D_{i,0} \approx D_{i,1} \approx D_{i+1}$$

It follows that $D_i \approx D_{i+1}$ for all $0 \leq i \leq l-1$, and we can conclude that

$$\llbracket E \rrbracket_f^{\Pi,\mathcal{SS}} = D_0 \approx D_l = \llbracket \mathsf{pattern}(E) \rrbracket_f^{\Pi,\mathcal{SS}}$$

Consider the patterns $E_{i,0}$ and $E_{i,1}$ defined as follows:

- $E_{i,0}$ is obtained from E_i by replacing each occurrence of a key share symbol K_{i+1}^j with a fresh key share symbol K^j, different from those in $\mathsf{KeyShares} \cup \{K_0^j \mid j \in 1..n\}$;
- $E_{i,1}$ is obtained from $E_{i,0}$ by replacing each occurrence of a subexpression $\{E\}_{K_{i+1}}$ with $\{\mathsf{struct}(E)\}_{K_{i+1}}$.

It follows from the description above that E_{i+1} can be obtained from $E_{i,1}$ by replacing each occurrence of a key share K^j with K_{i+1}^j.

Next we associate distributions (and therefore distribution ensembles) with the patterns $E_{i,0}$ and $E_{i,1}$ via a slight modification of the algorithm of Section 4. Specifically, we introduce computational interpretations for the newly introduced symbols K^1, K^2, $\ldots$, K^n: we add one new position to the array τ and one extra row to the matrix ϕ and set:

$$\tau[m+1] \xleftarrow{R} \mathcal{K}(\eta)$$

and

$$\phi[m+1] \xleftarrow{R} \mathcal{S}(\tau[m+1])$$

We use the bit-strings contained in $\phi[m+1]$ as interpretations of the symbols K^1, K^2, $\ldots$, K^n, so we add to the algorithm in Figure 3 the line:

$$\llbracket K^j \rrbracket_f^{\Pi,\mathcal{SS},\eta} = \phi[m+1][j]$$

For each $0 \leq i \leq l-1$, we let $D_{i,0}$ and $D_{i,1}$ be the distribution ensembles associated with $E_{i,0}$ and $E_{i,1}$.

The remainder of the proof consists of three steps. They respective establish that $D_i \approx D_{i,0}$, that $D_{i,0} \approx D_{i,1}$, and that $D_{i,1} \approx D_{i+1}$.

Step 1 ($D_i \approx D_{i,0}$)

The proof is by reduction: given an index $0 \leq h \leq l-1$ and an algorithm $\mathcal{A}$ such that $\mathbf{Adv}_{D_h,D_{h,0}}^{\mathrm{dist}}(\mathcal{A}, \eta)$ is non-negligible, we construct an adversary $\mathcal{B}$ against $\mathcal{SS}$ such that $\mathbf{Adv}_{\mathcal{SS}}^{ss}(\mathcal{B}, \eta)$ is also non-negligible. Therefore, the scheme $\mathcal{SS}$ is not a secure secret sharing scheme.

In the construction of $\mathcal{B}$ we use the set

$$S_h = \{j \mid K_{h+1}^j \text{ occurs in } E_h\}$$

of indexes j for which the key share K^j_{h+1} occurs in E_h. The occurrences of K^j_{h+1} in question cannot be under hidden keys, because of the acyclicity hypothesis. Therefore, crucially, S_h is a proper subset of $\{1, 2, \ldots, n\}$, for otherwise the key K_{h+1} would not be in $\mathsf{hidden}(E)$.

Let k_0 and k_1 be two arbitrary encryption keys generated via $k_0, k_1 \overset{R}{\leftarrow} \mathcal{K}(\eta)$. The adversary $\mathcal{B}$ that we construct receives as input a set of shares $(s_1, s_2, \ldots, s_t)$ sampled from one of the two distributions $\mathcal{S}(k_0)|_{S_h}$ or $\mathcal{S}(k_1)|_{S_h}$. It constructs a string s so that: if the input of $\mathcal{B}$ is sampled according to the distribution $\mathcal{S}(k_0)|_{S_h}$ then s is sampled according to D_h, and if the input of $\mathcal{B}$ is sampled according to the distribution $\mathcal{S}(k_1)|_{S_h}$ then s is sampled according to $D_{h,0}$. Adversary $\mathcal{B}$ then invokes the algorithm $\mathcal{A}$ on input s and outputs whatever $\mathcal{A}$ outputs. We therefore obtain that:

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{ss}}_{\mathcal{SS}}(\mathcal{B}, \eta) \\
&= \Pr[sh(k_0) \overset{R}{\leftarrow} \mathcal{S}(k_0, \eta) : \mathcal{B}(sh(k_0)|_{S_h}) = 1)] - \\
&\quad \Pr[sh(k_1) \overset{R}{\leftarrow} \mathcal{S}(k_1, \eta) : \mathcal{B}(sh(k_1)|_{S_h}) = 1)] \\
&= \Pr\left[s \overset{R}{\leftarrow} [\![E_h]\!]^{\Pi, \mathcal{SS}, \eta}_f : \mathcal{A}(s, \eta) = 1\right] - \\
&\quad \Pr\left[s \overset{R}{\leftarrow} [\![E_{h,0}]\!]^{\Pi, \mathcal{SS}, \eta}_f : \mathcal{A}(s, \eta) = 1\right] \\
&= \mathbf{Adv}^{\mathrm{dist}}_{D_h, D_{h,0}}(\mathcal{A}, \eta)
\end{aligned}$$

It follows that if the advantage of $\mathcal{A}$ is non-negligible then so is that of $\mathcal{B}$.

Adversary $\mathcal{B}$ computes s by applying to E_h a variant of the algorithm given in Section 4 for mapping expressions to bit-strings. Specifically, $\mathcal{B}$ starts by generating the vector τ of keys in which the entries $\tau[h+1]$ and $\tau[m+1]$ are set to k_0 and k_1. Next, $\mathcal{B}$ computes the entries in the matrix ϕ (used for defining the semantics of the key shares that occur in E_h). Both τ and ϕ have the appropriate distributions since k_0 and k_1 are randomly generated keys.

Then $\mathcal{B}$ computes the string s by recursively associating bit-strings with the subexpressions of E_h. The only departure from the algorithm of Section 4 is as follows. The bit-strings associated with the shares of K_{h+1} are interpreted using the input to $\mathcal{B}$. (In the original algorithm, the bit-string interpretation of these shares are the entries in $\phi[h+1]$, that is, shares of $\tau[h+1]$.) So, if $S_h = \{j_1, j_2, \ldots, j_t\}$, then the modified algorithm maps K^p_{h+1} to s_p, for each $0 \le p \le t$. Crucially, if the input to $\mathcal{B}$ is sampled according to $sh(k_0)|_{S_h}$, then the key K_{h+1} is mapped to k_0 and all shares $K^{j_p}_{h+1}$ are mapped to appropriate shares of k_0. Therefore, the string s is selected according to the distribution $[\![E_h]\!]^{\Pi, \mathcal{SS}, \eta}_f$. On the other hand, if the input to $\mathcal{B}$ is sampled according to $sh(k_1)|_{S_h}$, then although K_{h+1} is mapped to k_0, all its shares are mapped to shares of k_1—that is, to shares of a different key. Therefore, in this case, s is selected according to the distribution $[\![E_{h,0}]\!]^{\Pi, \mathcal{SS}, \eta}_f$, as desired.

Step 2 ($D_{i,0} \approx D_{i,1}$)

The proof is again by reduction: given an index $0 \le h \le l-1$ and an algorithm $\mathcal{A}$ that distinguishes between $D_{h,0}$ and $D_{h,1}$ with non-negligible probability, we construct an adversary $\mathcal{B}$ against Π such that $\mathbf{Adv}^{\mathrm{ind-cpa}}_{\Pi}(\mathcal{B}, \eta)$ is also non-negligible. The adversary $\mathcal{B}$ has access to the oracle $LR_{\Pi,b}$ and constructs a bit-string s that is sampled according to the distribution $D_{h,b}$. Then $\mathcal{B}$ invokes adversary $\mathcal{A}$ on s and outputs whatever $\mathcal{A}$ outputs. We thus obtain:

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{ind-cpa}}_{\Pi}(\mathcal{B}, \eta) \\
&= \Pr[\mathcal{B}^{LR_{\Pi,0}}(\eta) = 1] - \\
&\quad \Pr[\mathcal{B}^{LR_{\Pi,1}}(\eta) = 1] \\
&= \Pr[s \overset{R}{\leftarrow} D_{h,0} : \mathcal{A}(s) = 1] - \\
&\quad \Pr[s \overset{R}{\leftarrow} D_{h,1} : \mathcal{A}(s) = 1] \\
&= \mathbf{Adv}^{\mathrm{dist}}_{D_{h,0}, D_{h,1}}(\mathcal{A}, \eta)
\end{aligned}$$

It follows that if the advantage of $\mathcal{A}$ is non-negligible then so is that of $\mathcal{B}$. Therefore, the scheme Π is not an $\mathsf{IND\text{-}CPA}$ secure encryption scheme.

Adversary $\mathcal{B}$ computes s by applying to $E_{h,0}$ a variant of the algorithm given in Section 4. Specifically, $\mathcal{B}$ generates the vector of keys τ and the matrix of key shares ϕ and then recursively maps each subexpression F of $E_{h,0}$ to a bit-string. The interpretations of all basic symbols with the exception of the key K_{h+1} are as in Section 4. The interpretation of K_{h+1} is set to k, the key of the oracle. Since $\mathcal{B}$ does not actually have k, it is crucial that no share of K_{h+1} occurs in $E_{h,0}$. By acyclicity, there are no plain occurrences of K_{h+1} in $E_{h,0}$ either. There may however be uses of K_{h+1} as an encryption key. In order to deal with those occurrences, $\mathcal{B}$ makes uses of the oracle for producing encryptions under k, as follows.

Suppose that $E_{h,0}$ has a subexpression F of the form $\{E'\}_{K_{h+1}}$. First, $\mathcal{B}$ samples strings m_0 and m_1 from the distributions associated with E' and $\mathsf{struct}(E')$, respectively. This task is mostly straightforward since $\mathcal{B}$ knows the interpretations of all symbols that occur in E' and $\mathsf{struct}(E')$ with the exception of the interpretation of K_{h+1}. Whenever $\mathcal{B}$ needs to compute an encryption of a plaintext m under the key k (which is the interpretation of K_{h+1}), $\mathcal{B}$ submits to the oracle the pair (m, m) and obtains in return one such encryption. After producing m_0 and m_1 as described above, $\mathcal{B}$ submits to the oracle the pair (m_0, m_1) and sets c to be the answer returned by the oracle. Therefore, if the selection bit of the oracle is 0, then c is an encryption of m_0 under k, so c is distributed according to $[\![\{E'\}_{K_{h+1}}]\!]^{\Pi, \mathcal{SS}, \eta}_f$. If the selection bit of the oracle is 1, then c is an encryption of m_1 under k, so c is distributed according to $[\![\{\mathsf{struct}(E')\}_{K_{h+1}}]\!]^{\Pi, \mathcal{SS}, \eta}_f$.

Since $E_{i,1}$ is obtained by replacing in $E_{i,0}$ each subexpression F of the form $\{E'\}_{K_{i+1}}$ with $\{\mathsf{struct}(E')\}_{K_{i+1}}$, the overall result of $\mathcal{B}$'s computation is distributed like the interpretation of $E_{i,b}$.

Importantly, $\mathcal{B}$ is a valid $\mathsf{IND\text{-}CPA}$ adversary: each query (m_0, m_1) that $\mathcal{B}$ makes to its right-left oracle is valid since strings m_0 and m_1 are sampled according to distributions $[\![E']\!]^{\Pi, \mathcal{SS}, \eta}$ and $[\![\mathsf{struct}(E')]\!]^{\Pi, \mathcal{SS}, \eta}$, respectively, and therefore have equal lengths (by the assumption on the implementation discussed in Section 4).

Step 3 ($D_{i,1} \approx D_{i+1}$)

This step is similar to the proof that $D_i \approx D_{i,0}$: one can think of $E_{i,1}$ as obtained from E_{i+1} by replacing the key share symbols K^j_{i+1} with fresh key share symbols K^j, that is, in precisely the same manner in which $E_{i,0}$ is obtained from E_i. The same proof method applies.