Towards Privacy-Preserving XML Transformation

Meiko Jensen
Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany
meiko.jensen@rub.de

Florian Kerschbaum
SAP AG, SAP Research Center Karlsruhe
Vincenz-Prießnitz-Str. 1, 76131 Karlsruhe, Germany
florian.kerschbaum@sap.com

Abstract

In composite web services one can only either hide the identities of the participants or provide end-to-end confidentiality via encryption. For a designer of inter-organizational business processes this implies that she either needs to reveal her suppliers or force her customers to reveal their information. In this paper we present a solution to the encrypted data modification problem and reconciliate this apparent conflict. Using a generic sender-transformer-recipient example scenario, we illustrate the steps required for applying XML transformations to encrypted data, present the cryptographic building blocks, and give an outlook on advantages and weaknesses of the proposed encryption scheme. The transformer is then able to offer composite services without itself learning the content of the messages.

1. Introduction

In inter-organizational business processes, it is highly critical to protect the business partners' trade secrets, such as supplier indentities. Additionally, business processes tend to deal with sensitive information that requires best-possible data protection. When considering both requirements in conjunction, a new type of security problem occurs, which consist in the need to modify encrypted data in transit. As of today, the cryptographic technique of *homomorphic encryption* can be used to serve this need, however, full homomorphic encryption (e.g. [8]) is far from applicability to real-world scenarios due to its complexity.

In this paper, we present a slightly different solution approach to this important problem. Our approach uses a combination of *Goldwasser-Micali* [10] and *Sander-Young-Yung* [16] encryption schemes to provide a limited set of XML transformation operations that can be applied to encrypted XML documents, without revealing the contained data values themselves. Our solution is proven to be secure and shown to be feasible for real-world business processes as of today.

The paper is organized as follows. The next section gives an example scenario using the notion of sender-transformer-recipient, and derives the string operations that are necessary for performing XML transformations on encrypted data. Then, the basic encryption schemes are presented in Section 3, and the resulting cryptographic protocols for the string operations are given in Section 4. The paper concludes with related work in Section 5 and future research indications in Section 6.

2. Problem Statement

This section outlines the real-world environment we assumed for our approach, which is an inter-organizational business process setting that uses Web Services technology to realize a service-oriented architecture.

2.1. Service-Orientation

The *paradigm of service-orientation* consists in modularizing a distributed system architecture into a dedicated set of *services* that provide specific subtasks of the overall application. These *atomic* services can then be aggregated into *composite* services, which again can be used in other composite services, and so on. However, a crucial part of the paradigm is that a service user need not know the technical details of how a certain service is provided (e.g. whether it relies on the use of other, succeeding services or not), and vice versa a service provider does not necessarily know the context of why it actually gets invoked (e.g. whether the service users are regular clients or part of a composite service themselves).

Based on a generic example scenario, we show how the real-world problem can be mapped into a model that can be coped with in terms of cryptologic analysis.

2.2. Example Scenario

Assume a data sender to communicate with a data recipient via an intermediate data transformer. Assume the data transformer to have a vital interest in not revealing the data recipient's identity to the data sender, and vice versa. Assume all participants to agree to take part in a privacy-preserving communication protocol that prevents all but the data sender and the ultimate data recipient from getting to know the data sent. Further, assume the data recipient to prescribe a specific XML data format for incoming encrypted data, which may reveal the data recipient's identity. A solution has to enable a confidential transmission of an XML data fragment from the data sender to the data recipient such that none of the given assumptions is broken.

```
<Payment xmlns="http://resellerCorp.com">
<CCName>John Doe</CCName>
<CCType>VISA</CCType>
<CCNo>1111 2222 3333</CCNo>
<ExpirationDate>
<Month>04</Month>
<Year>12</Year>
</ExpirationDate>
</Payment>
```

Figure 1. Credit card data XML notation example

In order to get to a viable solution, we have to determine what basic operations are necessary to transform arbitrary credit card data representations (e.g. the one shown in Figure 1) into the recipient's representation. Commonly, such a transformation would be done using XML transformation languages like XSLT [13]. However, as we are dealing with encrypted data, there is no way to "parse" the encrypted XML, hence all of the common transformation approaches would fail. Nevertheless, the same transformation can be expressed using a limited set of string operations. For the given example, it would be sufficient to replace the string of `<Payment xmlns="http://resellerCorp.com">` with its pendant `<ccData xmlns="http://creditcorp.com">`. Accordingly, the `</Payment>` string is to be replaced with `</ccData>` in order to finish this first transformation step. We emphasize that this replacement may not be length preserving and that replacement of deterministically encrypted strings or characters is insufficient for ensuring privacy.

Most kinds of such XML transformations can be expressed by using cleverly crafted string replacements. However, it is important to note that all of these string operations substitute a known substring with another known substring, but never replace parts of the (unknown) data values. Hence, a string replacement can be done without any knowledge on the real data values contained in an encrypted XML data fragment. Thus, the solution to the encrypted data modification problem can be reduced to a solution of the string replacement problem for known substrings of an encrypted string (i.e. replacing known XML tags with other known XML tags).

2.3. String Operations

A general flaw of the described approach is that replacement of known substrings cannot be used for all kinds of XML transformation. For instance, there is no problem with replacing `<Month>` and `<Year>` of Figure 1 (and their closing tags) with the empty string `""` in order to move their contents to the XML parent element, but the opposite transformation is impossible without knowledge of the data values themselves (at least for the case of "splitting" or "moving" data values). Hence, the question arises on what classes of XML transformations require what kinds of string operations to be performed. For instance, changing the sequential order of the `CCName` and `CCNo` elements of Figure 1 cannot be expressed by pure string replacements. Hence, we accompanied the string replacement operation with three other basic string operations[1], namely *substring search*, *substring extraction*, and *string concatenation*. These basic operations are sufficient to support about all data-independent XML transformations imaginable. In the following, we present the most basic XML transformations, and show how they can be performed using only the given set of string operations.

2.4. XML Transformations on Encrypted Data

Based on the four basic string operations of *string replacement*, *substring search*, *substring extraction*, and *string concatenation*, a large set of XML transformations can be performed on encrypted XML fragments as well. However, due to the given restrictions

1. Obviously, the *string replacement* operation can be expressed using a combination of the other three operations. However, we keep it as a basic operation for convenience.

on transformations of XML text node values, it is not possible to perform all kinds of XML transformations with the given set of operations. For instance, some XML transformation languages like XSLT allow for testing whether an XML element's content value is smaller than a given number (if it is of integer value), and perform different transformations based on that test's result. Such a differentiation is not possible using the defined approach, since the test can not be performed on the encrypted data. However, it may be doubted on whether this is a real drawback, since enabling such comparisons would inevitably render any encryption scheme insecure due to the possibility of binary search on encrypted data.

In the following, we illustrate the set of possible XML transformations along with a description of the substring operations to be performed for them. Note that all of the following algorithms rely on a certain level of syntactical uniformity in the given XML fragment. This can be achieved by applying appropriate XML canonicalization algorithms (e.g. [3]) prior to encryption at the sender side.

2.4.1. Element and Text Replacement, Removal, and Concatenation. Renaming XML elements in an encrypted XML fragment can be achieved by straightforward use of the *string replacement* operator. Replacing all of the element's opening and closing tag names (i.e. the strings `"<X"` and `"</X>"`) with their new element names is sufficient here. Note that the opening tag should be given without its closing `">"` character, since it may contain unknown XML attributes or namespace declarations.

Element removal can be achieved by replacing both opening and closing tag with the empty string `""`. This way, potential child elements and text values are moved up to the parent element. This may result in text concatenation of sibling element's contents. For instance, removing `<Month>` and `<Year>` of Figure 1 would result in `"<ExpirationDate>04 12</ExpirationDate>"`. Depending on the XML Schema, it might be an option to e.g. replace the opening tag of `<Year>` with `"/"` instead of `""` to achieve a proper syntax for expiration date values. However, this largely depends on the data values' semantics.

The same replacement operations work for editing XML attribute names and known text values, however, in such cases it is required to verify that the string values only occur within the intended positions. For instance, using plain string replacement for renaming an attribute `foo` to `bar` will also result in replacement of every occurrence of `foo` with `bar` in all XML element tags, namespace declarations, and text node contents as well. This does not apply to XML elements, since the `"<"` character in XML is reserved explicitly for XML tag names.

2.4.2. Element and Text Insertion. If the encrypted XML fragment contains an empty element, it is possible to insert arbitrary child elements or text nodes. This can be achieved by replacing the string `"<X></X>"` with `"<X>"`+*fragment to insert*+`"</X>"`. Similarly, it is possible to insert arbitrary contents before or after a known opening or closing tag, e.g. replacing `"<X>"` with `"<X>"`+*fragment to insert*.

2.4.3. Element Swapping. In order to swap two adjacent elements `"<A>..</A>"` and `"<B>..</B>"`, the following algorithm can be applied. First, the document's prefix string s_{prefix} and fragment start position i_{start} are defined by searching for `"<A"`. Then, the document remainder after `"</B>"` called $s_{postfix}$ and its start position i_{end} are calculated using a search for `"</B>"` (with $i_{end} := search(\texttt{"</B>"}) + length(\texttt{"</B>"})$). Third, the intermediate index i_{middle} is calculated as i_{start} of a search for `"<B"`. The substrings of `<A>..</A>` and `<B>..</B>` are then defined as $s_A := substring(i_{start}, i_{middle})$ and $s_B := substring(i_{middle}, i_{end})$, respectively. The result of the swap operation is then defined by concatenating $s_{prefix} + s_B + s_A + s_{postfix}$.

3. Building Blocks

In order to realize the described XML element transformations in a privacy-protecting way, the following basic cryptographic protocols are utilized.

3.1. Goldwasser-Micali Encryption

Goldwasser-Micali (GM) encryption [10] is a public-key, semantically-secure, homomorphic encryption scheme. Its plaintext length is 1 bit and its security relies on the fact, that one cannot compute the Jacobi symbol without factorization. Note, that the computation of a Jacobi symbol is a cheap operation compared to modular exponentiation and therefore the encryption is quite fast albeit operating on bits.

Let $E_A(x)$ denote encryption of x under Alice's GM public key. Multiplying two ciphertexts, e.g. $E_A(x) \cdot E_A(y)$, results in an encryption of the exclusive-or (XOR) denoted by $\oplus$. Obviously this allows negation (XOR 1), but it also allows AND with an unencrypted bit. To achieve the AND operation of $E_A(x)$ and an unencrypted bit y, choose $E_A(y)$ if $y = 0$ and choose

$E_A(x)$ if $y = 1$. We denote this operation as $\circ$, e.g. $E_A(x) \circ y$. In summary,

$$E_A(x) \cdot E_A(y) = E_A(x \oplus y)$$
$$E_A(x) \circ y = E_A(x \odot y)$$

GM encryption is randomized (and semantically-secure), i.e. one cannot infer from the ciphertext and the public key whether it is a specific plaintext. Randomization and homomorphism implies the ability to re-randomize: simply XOR $E_A(0)$. This allows encryptions to be untraceable, i.e. the result of an operation on an encrypted value is uniformly distributed among the ciphertexts (of that plaintext).

3.2. Sander-Young-Yung Encryption

Sander-Young-Yung (SYY) encryption [16] is also a public-key, semantically-secure, homomorphic encryption scheme. It probabilistically encrypts a bit, i.e. there is a small probability of decrypting a 0 as a 1. There is a security parameter λ and the probability of incorrectly decrypting a 0 is $2^{-\lambda}$.

SYY encryption bases its security, semantic security property, and re-randomization on the GM encryption, but the homomorphic operation is different. Let $E^{\star}_A(x)$ denote encryption of x under Alice's SYY public key. The element-wise product $E^{\star}_A(x) \times E^{\star}_A(y)$ of two encryptions $E^{\star}_A(x)$ and $E^{\star}_A(y)$ is the logical and (AND – $\odot$) of the two plaintexts:

$$E^{\star}_A(x) \times E^{\star}_A(y) = E^{\star}_A(x \odot y)$$

A bit encrypted in GM encryption $E_A(b)$ can be converted into the same bit encrypted in SYY encryption $E^{\star}_A(b)$ with a simple procedure. The reverse conversion is not possible. We refer the reader to [16] for details.

3.2.1. Oblivious Transfer. In Oblivious Transfer (OT) Alice has two inputs a_0 and a_1 and Bob has a bit b. At the end of the protocol, Bob will have obtained a_b, but learnt nothing about $a_{\neg b}$ and Alice has not learnt b. As of today, the fastest implementation of OT is described in [15].

3.2.2. Yao's Millionaires' Protocol. Yao's millionaires' problem was introduced (and solved) along with the problem of secure computation [19]. Two millionaires want to compare their wealth, but do not want to reveal the exact amount to the other party. I.e., Alice has a and Bob has b and they want to compute $a < b$ without revealing a to Bob and b to Alice. This can be achieved using the general circuit construction technique [19]. Furthermore, the problem of comparison has received some attention of its own, e.g. [6], and clever faster solutions have been found.

An extension of Yao's millionaires' problem is a split-result Yao's millionaires' problem. In this case, again Alice has a and Bob b, but they compute $a < b$, such that the result is shared in a secret sharing scheme, i.e. Alice obtains a random bit r and Bob obtains $r \oplus (a < b)$. Neither Alice nor Bob alone gain information about the comparison, but they can use that value in future computations. A protocol can be constructed using the circuit construction technique by having Alice submitting a random bit as input and returning the XOR of that bit and the result of the regular protocol to Bob only, or special protocols may be able to be modified for this purpose.

4. Protocols for Basic Operations

4.1. String Representation

Each string is represented as a length field and sequence of characters. The sequence is padded with 0s to the maximum length of l_{max} characters. We recommend to align the maximum length with byte lengths of the length field, i.e. 255 byte strings (1 byte length field), 64 Kbyte - 1 (2 byte length field), 4GByte - 1 (4 byte length field), etc. In the following, let $\mathbf{L}$ be a string, i.e. $\mathbf{L} = \{l, \vec{x}\}$ where l denotes the length and $\vec{x}$ is the padded character sequence.

4.2. Providing Input

Our protocols are two-party protocols on shared strings, i.e. we assume that the parties use these secret shares as input.

In order to provide (known) input $\vec{x}$, e.g. an XML tag, Alice, e.g. the transformer, creates a random share by uniformly choosing a random number r' ($0 \leq r' \leq l_{max}$) and (also uniformly) a random vector $\vec{r''}$ (of length l_{max}). She sends $\mathbf{L}_B = \{r', \vec{r''}\}$ to Bob, e.g. the recipient, and sets her own string $\mathbf{L}_A$ to $\{l - r' \bmod l_{max}, \vec{x} \oplus \vec{r''}\}$.

In order to send a message, Charlie, e.g. the sender, similarly splits his input into $\mathbf{L}_A$ and $\mathbf{L}_B$, but instead of keeping his share $\mathbf{L}_A$ he encrypts it with Alice's public key. Furthermore he encrypts $\mathbf{L}_B$ with Bob's public key and sends both ciphertexts to Alice. Alice can decrypt her share $\mathbf{L}_A$ and forwards the encrypted share of Bob. After decryption Alice and Bob can then execute the protocols for the basic string operations. Note that Alice can prevent Charlie from learning Bob's identity (and needing to use his public key) by using proxy-reencryption [1].

4.3. String Concatenation

The string concatenation protocol composes a shared string based on two other shared strings $\mathbf{X}$ and $\mathbf{Y}$. After executing the protocol the two parties have a share of the concatenated string $\mathbf{Z}$.

The basic idea of the protocol is to shift one string by the length of the other string (to the right) and then combine them. The combination can be done with XOR, since after shifting each character is partnered with 0-padding character. There is one problem with shifting: The length is shared modulo l_{max}, i.e. it may be larger than l_{max}. Then shifting would move to an incorrect position, but if we rotate (to the right) instead of shift, the problem is fixed. The complete protocol proceeds as follows:

Input: Shared strings $\mathbf{X} = \{l, \vec{x}\}$ and $\mathbf{Y} = \{m, \vec{y}\}$, i.e. Alice has $\mathbf{X}_A = \{l_A, \vec{x}_A\}$, $\mathbf{Y}_A = \{m_A, \vec{y}_A\}$ and Bob has $\mathbf{X}_B = \{l_B, \vec{x}_B\}$, $\mathbf{Y}_B = \{m_B, \vec{y}_B\}$ where $l = l_A + l_B \bmod l_{max}$, $\vec{x} = \vec{x}_A \oplus \vec{x}_B$, $m = m_A + m_B \bmod l_{max}$, $\vec{y} = \vec{y}_A \oplus \vec{y}_B$.

Output: Shared string $\mathbf{Z} = \{n, \vec{z}\}$, i.e. Alice obtains $\mathbf{Z}_A = \{n_A, \vec{z}_A\}$ and Bob obtains $\mathbf{Z}_B = \{n_B, \vec{z}_B\}$, such that $n = n_A + n_B \bmod l_{max}$, $\vec{z} = \vec{z}_A \oplus \vec{z}_B$ and $\mathbf{Z}$ is the concatenation of $\mathbf{X}$ and $\mathbf{Y}$.

1) Alice chooses (uniformly) a random vector $\vec{r}_A$ (of length l_{max}). She sets $\vec{y}'_A = \vec{y}_A \oplus \vec{r}_A$ and sends $E_A(\vec{y}'_A)$ to Bob.
2) Bob chooses (uniformly) a random vector $\vec{r}_B$ (of length l_{max}). He computes $\vec{e} = E_A(\vec{y}'_A \oplus \vec{r}_B) = E_A(\vec{y}'_A) \cdot E_A(\vec{r}_B)$. He rotates to the right $\vec{e}$ by l_B characters: $\vec{e}' = rotate\_right_{8l_B}(\vec{e})$, i.e. $e'_i = e_{i-8l_B \bmod 8l_{max}}$. He sets $\vec{y}'_B = \vec{y}_B \oplus \vec{r}_B$. He sends $E_B(\vec{y}'_B)$ and $\vec{e}'$ to Alice.
3) Alice decrypts $\vec{e}'$: $\vec{d} = D_A(\vec{e}')$. She rotates to the right $\vec{d}$ by l_A characters: $\vec{d}' = rotate\_right_{8l_A}(\vec{d})$. She sets her share of $\mathbf{Z}$ to $\mathbf{Z}_A = \{l_A + m_A, \vec{x}_A \oplus \vec{d}'\}$. She computes $\vec{g} = E_B(\vec{y}'_B \oplus \vec{r}_A) = E_B(\vec{y}'_B) \cdot E_B(\vec{r}_A)$. She rotates to the right $\vec{g}$ by l_A characters: $\vec{g}' = rotate\_right_{8l_A}(\vec{g})$. She sends $\vec{g}'$ to Bob.
4) Bob decrypts $\vec{g}'$: $\vec{f} = D_B(\vec{g}')$. He rotates to the right $\vec{f}$ by l_B characters: $\vec{f}' = rotate\_right_{8l_B}(\vec{f})$. He sets his share of $\mathbf{Z}$ to $\mathbf{Z}_B = \{l_B + m_B, \vec{x}_B \oplus \vec{f}'\}$.

4.3.1. Security Proof. The proof of security for a secure computation is by comparison to the ideal model. In the ideal model both parties send their input to a trusted third party and receive the result from it. We prove security in the semi-honest model [9]. In the semi-honest model the parties follow the protocol as prescribed, but keep a record of all messages and try to infer as much information as possible. Therefore we have to show that every message received can be computed by a simulator given input and output of the protocol, i.e. the output of the simulator is computationally indistinguishable from Alice's view of the protocol. Our proof relies on a cryptographic theorem.

First, that the encryption of a bit is (semantically) secure, i.e. the probability of guessing the plaintext given the cipher text and the public key is like a random guess (i.e. it is at most negligible better than that).

$$P[m = b | E_X(m), E_X(\cdot)] \leq \frac{1}{2} + \frac{1}{poly(|E_X(\cdot)|)} \quad (1)$$

Proof: We first construct a simulator for Alice's view. Recall that Alice's input is $\mathbf{X}_A = \{l_A, \vec{x}_A\}$ and $\mathbf{Y}_A = \{m_A, \vec{y}_A\}$ and her output is $\mathbf{Z}_A = \{n_A, \vec{z}_A\}$.

1) Uniformly choose a random sequence $\vec{r}_1$ of l_{max} characters. Set $E_B(\vec{y}'_B) = E_B(\vec{r}_1)$.
2) Rotate $\vec{z}_A \oplus \vec{x}_A$ to the left by l_A characters: $\vec{d} = rotate\_left_{8l_A}(\vec{z}_A \oplus \vec{x}_A)$ Set $\vec{e}' = E_A(\vec{d})$.

We now prove computationally indistinguishability. The computational indistinguishability of step 1 is a corollary of (1). For step 2 the corresponding message is derived from the output. We omit the proof for Bob's view for brevity, since it is the mirror image of Alice's view. □

4.4. Substring

The composite protocol for substring algorithm for generating a shared substring $\mathbf{Z}$ uses a shared string $\mathbf{X}$, a shared index s and a shared length t as input. This algorithm consists of two basic algorithms, the substring protocol with remainder bytes and the masking protocol for remainder bytes. The first protocol rotates the shared string $\mathbf{X}$ to the left by s. This protocol is similar to the concatenation protocol and proceeds as follows:

Input: Shared string $\mathbf{X} = \{l, \vec{x}\}$, i.e. Alice has $\mathbf{X}_A = \{l_A, \vec{x}_A\}$ and Bob has $\mathbf{X}_B = \{l_B, \vec{x}_B\}$ where $l = l_A + l_B \bmod l_{max}$, $\vec{x} = \vec{x}_A \oplus \vec{x}_B$. Shared index s and length t, i.e. Alice has s_A and t_A and Bob has s_B and t_B where $s = s_A + s_B \bmod l_{max}$ and $t = t_A + t_B \bmod l_{max}$.

Output: Shared string $\mathbf{Z} = \{n, \vec{z}\}$, i.e. Alice obtains $\mathbf{Z}_A = \{n_A, \vec{z}_A\}$ and Bob obtains $\mathbf{Z}_B = \{n_B, \vec{z}_B\}$, such that $n = n_A + n_B \bmod l_{max} = t$, $\vec{z} = \vec{z}_A \oplus \vec{z}_B$ and the $(0, t)$-substring of $\mathbf{Z}$ is (s, t)-substring of $\mathbf{X}$.

1) Alice chooses (uniformly) a random vector $\vec{r}_A$ (of length l_{max}). She sets $\vec{x}'_A = \vec{x}_A \oplus \vec{r}_A$ and sends $E_A(\vec{x}'_A)$ to Bob.

2) Bob chooses (uniformly) a random vector $\vec{r}_B$ (of length l_{max}). He computes $\vec{e} = E_A(\vec{x}'_A \oplus \vec{r}_B) = E_A(\vec{x}'_A) \cdot E_A(\vec{r}_B)$. He rotates to the left $\vec{e}$ by s_B characters: $\vec{e}' = rotate\_left_{8s_B}(\vec{e})$, i.e. $e'_i = e_{i+8s_B \bmod 8l_{max}}$. He sets $\vec{x}'_B = \vec{x}_B \oplus \vec{r}_B$. He sends $E_B(\vec{x}'_B)$ and $\vec{e}'$ to Alice.
3) Alice decrypts $\vec{e}'$: $\vec{d} = D_A(\vec{e}')$. She rotates to the left $\vec{d}$ by s_A characters: $\vec{d}' = rotate\_left_{8s_A}(\vec{d})$. She sets her share of $\mathbf{Z}$ to $\mathbf{Z}_A = \{t_A, \vec{d}'\}$. She computes $\vec{g} = E_B(\vec{x}'_B \oplus \vec{r}_A) = E_B(\vec{x}'_B) \cdot E_B(r_A)$. She rotates to the left $\vec{g}$ by s_A characters: $\vec{g}' = rotate\_left_{8s_A}(\vec{g})$. She sends $\vec{g}'$ to Bob.
4) Bob decrypts $\vec{g}'$: $\vec{f} = D_B(\vec{g}')$. He rotates to the left $\vec{f}$ by s_B characters: $\vec{f}' = rotate\_left_{8s_B}(\vec{f})$. He sets his share of $\mathbf{Z}$ to $\mathbf{Z}_B = \{t_B, \vec{f}'\}$.

The second protocol verifies the length of the substring and makes sure that it is filled with zeros after the position t, since the rotation may have left some remainder characters and it is important for the composition of several protocols that all strings are only padded with 0 characters. Its basic idea is to construct a masking string of the correct length and compute the AND operation of it and the string. This protocols proceeds as follows:

Input: Shared string $\mathbf{X} = \{l, \vec{x}\}$, i.e. Bob has $\mathbf{X}_B = \{l_B, \vec{x}_B\}$, but Alice only needs l_A where $l = l_A + l_B \bmod l_{max}$.

Output: "Shared share" of string $\mathbf{Z}$, i.e. Alice obtains $\vec{z}_A$ and Bob obtains $\vec{z}_B$, such that $\{l_A, \vec{z} = \vec{z}_A \oplus \vec{z}_B\}$ is a share of string $\mathbf{Z}$ that has all trailing characters set to 0.

1) Alice prepares a mask vector $\vec{m} = 1^{8l_A}0^{8l_{max}-8l_A}$ with $8l_A$ heading ones and trailing zeroes till l_{max} characters. She sends $\vec{e} = E_A(\vec{m})$ to Bob.
2) Bob prepends $8l_B$ ones and appends zeroes till $2l_{max}$ characters to $\vec{m}$, i.e. he prepares an encrypted mask vector $\vec{e}' = E_A(\vec{m}') = E_A(1^{8l_B}), E_A(\vec{m}), E_A(0^{8l_{max}-8l_B})$. Let $\vec{e}'_{head} = E_A(m'_{head})$ be the heading l_{max} characters and $\vec{e}'_{tail} = E_A(m'_{tail})$ be the trailing l_{max} characters of $\vec{e}'$, i.e. $\vec{e}'_{head} = e'_0, \ldots, e'_{8l_{max}-1}$ and $\vec{e}'_{tail} = e'_{8l_{max}}, \ldots, e'_{16l_{max}-1}$. Bob computes $\vec{e}'' = E_A(\vec{m}'') = \vec{e}'_{head} \cdot \vec{e}'_{tail} = E_A(\vec{m}'_{head} \oplus \vec{m}'_{tail})$. He prepares two vectors $\vec{e}''_0$ and $\vec{e}''_1$: one for the case $l_A + l_B \geq l_{max}$ and one for the case $l_A + l_B < l_{max}$. He sets $\vec{e}''_0 = E_A(\vec{m}''_0) = \vec{e}'' \cdot E_A(1^{8l_{max}}) = E_A(\vec{m}'' \oplus 1^{8l_{max}})$ and $\vec{e}''_1 = E_A(\vec{m}''_1) = \vec{e}''$. He computes both $\vec{e}'''_{0,1} = E_A(\vec{m}'''_{0,1}) = \vec{e}''_{0,1} \circ x_B = E_A(\vec{m}''_{0,1} \odot x_B)$. He chooses (uniformly) a random vector $\vec{r}_B$ (of length $8l_{max}$). He computes both $\vec{e}''''_{0,1} = E_A(\vec{m}''''_{0,1}) = \vec{e}'''_{0,1} \cdot E_A(r_B) = E_A(\vec{m}'''_{0,1} \oplus r_B)$.
3) Alice and Bob engage in a split-result Yao's millionaires' protocol for l_A and $l_{max} - l_B$. Alice obtains ρ_A and Bob obtains ρ_B and the bit $l_A + l_B < l_{max} = l_A < l_{max} - l_B = \rho_A \oplus \rho_B$.
4) Bob prepares two messages: $\vec{o}_0 = \vec{e}''''_{\rho_B}$ and $\vec{o}_1 = \vec{e}''''_{1\oplus\rho_B}$. Alice and Bob engage in an Oblivious Transfer and Alice obtains $\vec{f} = \vec{o}_{\rho_A} = \vec{e}''''_{\rho_A \oplus \rho_B}$.
5) Alice decrypts $\vec{f}$: $\vec{d} = D_A(\vec{f})$. She chooses (uniformly) a random vector $\vec{r}_A$ (of length $8l_{max}$). She computes $\vec{d}' = \vec{d} \oplus r_A$ and sets her "share of the share" $\vec{z}_A = \vec{r}_A$. She sends $\vec{d}'$ to Bob.
6) Bob computes $\vec{d}'' = \vec{d}' \oplus \vec{r}_B$ and sets his "share of the share" $\vec{z}_B = \vec{d}''$.

The following composite protocol uses the masking protocol twice after the basic substring protocol to clean both sides of the shared string.

Input: Shared string $\mathbf{X} = \{l, \vec{x}\}$, i.e. Alice has $\mathbf{X}_A = \{l_A, \vec{x}_A\}$ and Bob has $\mathbf{X}_B = \{l_B, \vec{x}_B\}$ where $l = l_A + l_B \bmod l_{max}$, $\vec{x} = \vec{x}_A \oplus \vec{x}_B$. Shared index s and length t, i.e. Alice has s_A and t_A and Bob has s_B and t_B where $s = s_A + s_B \bmod l_{max}$ and $t = t_A + t_B \bmod l_{max}$.

Output: Shared string $\mathbf{Z} = \{n, \vec{z}\}$, i.e. Alice obtains $\mathbf{Z}_A = \{n_A, \vec{z}_A\}$ and Bob obtains $\mathbf{Z}_B = \{n_B, \vec{z}_B\}$, such that $n = n_A + n_B \bmod l_{max}$, $\vec{z} = \vec{z}_A \oplus \vec{z}_B$ and $\mathbf{Z}$ is the (s,t)-substring of $\mathbf{X}$.

1) Alice and Bob engage in the "substring with remainder" protocol. Alice obtains $\mathbf{Z}'_A = \{n'_A, \vec{z}'_A\}$ and Bob obtains $\mathbf{Z}'_B = \{n'_B, \vec{z}'_B\}$.
2) Alice and Bob engage in the "masking" protocol with $\mathbf{Z}'$. Alice obtains $\vec{z}''_A$ and Bob $\vec{z}''_B$.
3) Alice and Bob engage in the "masking" protocol with $\mathbf{Z}'$ again, but this time with the roles of Alice and Bob interchanged. Alice obtains $\vec{z}'''_A$ and Bob $\vec{z}'''_B$.
4) Alice sets her share to $\mathbf{Z}_A = \{n'_A, \vec{z}''_A \oplus \vec{z}'''_A\}$. Bob sets his share to $\mathbf{Z}_B = \{n'_B, \vec{z}''_B \oplus \vec{z}'''_B\}$.

4.4.1. Security Proof. The security of the composite protocol follows directly from the composition theorem of [9], since it is such a composition. In order to complete the proof we need to prove the security of the "substring with remainder" and the "masking" protocol.

Substring Protocol with Remainder Bytes.

Proof: The simulator for Alice's view is:

1) Uniformly choose a random sequence $\vec{r}_1$ of l_{max} characters. Set $E_B(\vec{x}'_B) = E_B(\vec{r}_1)$.

2) Rotate $\vec{z}_A$ to the right by s_A characters: $\vec{d} = rotate\_right_{8s_A}(\vec{z}_A)$ Set $\vec{e} = E_A(\vec{d})$.

The simulator is almost identical to the simulator of the string concatenation protocol. Bob's view is again a mirror image and omitted for brevity. □

Masking Protocol. In the masking protocol we will replace the invocations to Yao's millionaires' protocol and Oblivious Transfer by the ideal functionality using Goldreich's composition theorem. The composition theorem for secure multi-party computation [9] states, loosely speaking, that a protocol using secure building blocks protocols is secure if it is secure when those are replaced by invocations to ideal trusted third party. The security of these building blocks has been established independently [15], [19].

Proof: The simulator for Alice's view is:

1) Uniformly choose a random bit r_1. Set $\rho_A = r_1$.
2) Uniformly choose a random vector $\vec{r}_2$ of l_{max} characters. Set $\vec{f} = E_A(\vec{z}_A \oplus \vec{r}_2)$.

Step 1 follows from the definition of the split Yao's millionaires' protocol. The security of the message in step 2 follows from the random-pad security, since $\vec{r}_B$ has been chosen uniformly and is used only to hide this message.

The simulator for Bob's view is:

1) Uniformly choose a random vector $\vec{r}_1$ of l_{max} characters. Set $\vec{e} = E_A(\vec{r}_1)$.
2) Uniformly choose random bit r_2. Set $\rho_B = r_2$.
3) Uniformly choose a random vector $\vec{r}_3$ of l_{max} characters. Set $\vec{d'} = \vec{z}_B \oplus \vec{r}_3$.

Step 1 follows from the security of encryption (see equation (1) in section 4.3.1). The distribution of step 3 follows from the definition of the split-result Yao's millionaires' protocol. The security of the message in step 3 follows from the security of the random-pad, since $\vec{r}_A$ has been chosen uniformly and used only to hide this message. □

4.5. String Search

The string search protocol searches a shared string $\mathbf{X}$ for a pattern string $\mathbf{P}$. In this protocol the output is known to both parties and not shared. After executing this protocol the two parties have knowledge of each position in $\mathbf{X}$ where $\mathbf{P}$ occurs.

The protocol uses the simple $O(nm)$ string search and computes the XOR of the two strings at each position. Then the result of the XOR is compressed by switching the encryption method, such that the overall communication complexity can be reduced significantly. The final protocol proceeds as follows:

Input: Shared string $\mathbf{X} = \{l, \vec{x}\}$, i.e. Alice has $\mathbf{X}_A = \{l_A, \vec{x}_A\}$ and Bob has $\mathbf{X}_B = \{l_B, \vec{x}_B\}$ where $l = l_A + l_B \bmod l_{max}$, $\vec{x} = \vec{x}_A \oplus \vec{x}_B$. Bob has a search pattern string $\mathbf{P} = \{q, \vec{p}\}$.

Output: Alice and optionally Bob have a binary vector $\vec{m}$ of length l_{max} where $m_i = 1$, if $\mathbf{P}$ occurs at position i in $\mathbf{X}$, and $m_i = 0$, otherwise.

1) Alice sends $E_A(\vec{x}_A)$ to Bob.
2) For each position i from 0 to $l_{max} - q$ Bob extracts the (encrypted) subsequence $\vec{e}_i = E_A(\vec{d}_i) = E_A(\vec{x}_A)_{8i}, \ldots, E_A(\vec{x}_A)_{8i+8q-1}$ from character i to character $i + q$ (inclusive) of $E_A(\vec{x}_A)$. He also extracts the same subsequence from his share $\vec{x}_B$ of $\mathbf{X}$: $\vec{h}_i = x_{B,8i}, \ldots, x_{B,8i+8q-1}$. He computes $\vec{g}_i = E_A(\vec{f}_i) = \vec{e}_i \cdot E_A(\vec{h}_i) \cdot E_A(\vec{p}) \cdot E_A(1^{8q}) = E_A(\vec{d}_i \oplus \vec{h}_i \oplus \vec{p} \oplus 1^{8q})$. He converts the Goldwasser-Micali encryption of each $\vec{g}_i$ into a Sander-Young-Yung encryption: $\vec{g}'_i = E^\star_A(\vec{f}_i)$. He computes the bit-wise AND of each $\vec{g}'_i$: $m'_i = E^\star_A(m_i) = g'_{i,1} \times \ldots \times g'_{i,8q} = E^\star_A(f_{i,1} \odot \ldots \odot f_{i,8q})$. He appends $q-1$ encrypted 0's to $\vec{m}'$: $\vec{m}'' = \vec{m}', E^\star_A(0^{q-1})$. He sends $\vec{m}''$ to Alice.
3) Alice decrypts $\vec{m}''$: $\vec{m} = D^\star_A(\vec{m}'')$ and sends this result to Bob.

4.5.1. Security Proof. *Proof:* The simulator for Alice's view is particularly simple and given without further explanation. Let $\vec{m}$ be the output vector.

1) Set $\vec{m}'' = E^\star_A(\vec{m})$.

The simulator for Bob's view is:

1) Uniformly choose a random vector $\vec{r}_1$ of l_{max} characters. Set $E_A(\vec{x}_A) = E_A(\vec{r}_1)$.

The distribution of step 1 is indistinguishable due to the security of the encryption (see equation (1) in Section 4.3.1). □

5. Related Work

Our privacy-preserving XML transformations represent special protocols for secure two-party computation (STPC). In STPC two parties have an input each and want to compute a joint function of their input without revealing anything but the result. STPC has been introduced in [19], but we improve over this general circuit construction technique as implemented in FairPlay [14]. All our protocols have communication complexity $O(l_{max})$ whereas STPC has $O(l^2_{max})$ due to the variable array indexing. Our string concatenation and substring protocols also improve running to $O(l_{max})$ compared to $O(l^2_{max})$.

We are not aware of any special secure protocols for our basic string operations. Searchable encryption, e.g. [2], [17], offers a non-interactive technique for searching for a string in a ciphertext. This functionality is somewhat limited for our purposes particularly with respect to composition. Privacy-preserving regular expression search is presented in [18].

Our protocols employ a number of cryptographic techniques including homomorphic encryption. Recently fully homomorphic encryption that can implement any function securely has been introduced [8]. We emphasize that our protocols are significantly more efficient than fully homomorphic encryption, but our protocols also leak additional information. For instance, they leak the number and order of applied basic operations. Note, that if this information may not be revealed to any party, the algorithm has to always run for the maximum number of basic operations possible. If the running time of the protocol depends on secret input, this input may be inferred via timing attacks. A secure protocol's average complexity therefore always equals its worst-case complexity. In case of XSLT this worst case complexity is exponential in the input length, and we therefore predict that even if efficient fully homomorphic encryption is found it cannot be applied to XSLT.

6. Conclusion and Future Work

In this paper we presented a solution to the problem of modifying encrypted data in inter-organizational service compositions in a privacy-preserving way. The solution is based on a new data encryption scheme that allows a large set of XML transformations to be applied to the encrypted data, without revealing the data values themselves. The scheme is useful in all scenarios where the data structure itself reveals information to the recipient that it should not get according to the interests of a data transformer that is not the data originator itself (e.g. a reseller).

The encryption scheme provides a strong level of confidentiality, however, some further issues have to be addressed. At first, there is no way yet for a data recipient to verify the integrity of the data received by the transformer. Hence, the transformer could maliciously add new data items—which would be indistinguishable from real original data—or re-use data fragments to craft additional messages, potentially resulting in an unintended revelation of the confidential data items to the transformer (e.g. by sending a data copy to some echoing service at the recipient). In order to address these issues, future work consists in analyzing the use of digital signatures in conjunction with the proposed encryption scheme in order to assure data integrity as well. Beyond that, an obvious future work will be an in-depth performance and security analysis of a real-world implementation of the proposed scheme.

References

[1] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. *Proceedings of the Network and Distributed System Security Symposium, p. 29-43*, 2005.

[2] D. Boneh, G. DiCrescenzo, R. Ostrovsky, and G. Persiano. Public-key Encryption with Keyword Search. *Proceedings of Eurocrypt, p. 506-522*, 2004.

[3] J. Boyer, D. Eastlake, and J. Reagle. Exclusive XML Canonicalization Version 1.0. *W3C Recommendation*, 2002.

[4] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, and Ed Simon. XML Encryption Syntax and Processing. *W3C Recommendation*, 2002.

[5] D. C. Fallside, and P. Walmsley. XML Schema Part 0: Primer Second Edition. *W3C Recommendation*, 2004.

[6] M. Fischlin. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. *RSA Security Cryptographer's Track, p. 457-472*, 2001.

[7] S. Gajek, M. Jensen, L. Liao, and J. Schwenk. Analysis of Signature Wrapping Attacks and Countermeasures. *Proceedings of the IEEE 7th International Conference on Web Services, p. 575-582*, 2009.

[8] C. Gentry. Fully Homomorphic Encryption using Ideal Lattices. *Proceedings of the 41st ACM Symposium on Theory of Computing, p. 169-178*, 2009.

[9] O. Goldreich. Secure Multi-party Computation. Available at *www.wisdom.weizmann.ac.il/˜oded/pp.html*, 2002.

[10] S. Goldwasser, and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Science 28(2), p. 270-299*, 1984.

[11] R. Herkenhöner, M. Jensen, H.C. Pöhls, and H. de Meer. Towards Automated Processing of the Right of Access in Inter-Organizational Web Service Compositions. *Proceedings of the IEEE 6th World Congress on Services, p. 645-652*, 2010.

[12] M. Jensen, and N. Gruschka. Privacy Against the Business Partner: Issues for Realizing End-to-End Confidentiality in Web Service Compositions. *Proceedings of the 20th International Workshop on Database and Expert Systems Applications, p. 117-121*, 2009.

[13] M. Kay. XSL Transformations (XSLT) Version 2.0. *W3C Recommendation*, 2007.

[14] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-party Computation System. *Proceedings of the 13th USENIX security symposium, p. 287-302*, 2004.

[15] M. Naor, and B. Pinkas. Efficient Oblivious Transfer Protocols. *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, p. 448-457*, 2001.

[16] T. Sander, A. Young, and M. Yung. Non-Interactive CryptoComputing for NC1. *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, p. 554-566*, 1999.

[17] D. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. *Proceedings of IEEE Symposium on Security and Privacy, p. 44-55*, 2000.

[18] J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy Preserving Error Resilient DNA Searching through Oblivious Automata. *Proceedings of the 14th ACM Conference on Computer and Communications Security, p. 519-528*, 2007.

[19] A. Yao. Protocols for Secure Computations. *Proceedings of the IEEE Symposium on foundations of computer science* 23, p. 160-164, 1982.