Sanitizable Signatures in XML Signature — Performance, Mixing Properties, and Revisiting the Property of Transparency*

Henrich C. Pöhls, Kai Samelin, Joachim Posegga

Chair of IT Security, University of Passau, Germany
(hp|ks|jp)@sec.uni-passau.de

Abstract. We present the performance measures of our Java Cryptography Architecture (JCA) implementation that integrates sanitizable signature schemes into the XML Signature Specification. Our implementation shows mostly negligible performance impacts when using the *Ateniese* scheme with four different chameleon hashes and the *Miyazaki* scheme in XML Signatures. Thus, sanitizable signatures can be added to the XML Security Toolbox. Applying the new tools we show how to combine different hash algorithms over different document parts adding and removing certain properties of the sanitizable signature scheme; this mixing comes very natural in XML Signatures. Finally, we motivate that existing definitions for the property of Transparency are counterintuitive in these combinations. Our conclusion is that the document-level Transparency property is independent of the sub-document properties Weak and Strong Transparency.

Keywords: Sanitizable Signatures, Transparency, Performance, XML Signature Framework

Note: This version is a slightly enhanced version of our paper that appeared in the proceedings of ACNS 2011 by J. Lopez and G. Tsudik (Eds.), LNCS 6715, pp. 166–182, 2011. Springer-Verlag Berlin Heidelberg 2011

1 Introduction

A sanitizable signature scheme (SSS) allows a defined third party, the so-called sanitizer, to alter an already signed document without invalidating the signature and without involving the original signer again. This comes in handy for many applications, examples thereof can be found in each scheme's literature. However, sanitizable signature schemes are not part of the proposed signature methods in the XML Signature Syntax and Processing W3C Standard [7][1]. We

* This research is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUe-IT project

[1] We will refer to this as XML Signature for brevity.

have implemented the sanitizable signature scheme proposed in 2005 by *Ateniese* [1] in Java. The *Ateniese* scheme is build upon a chameleon hash; we implemented chameleon hashes proposed by *Krawczyk* and *Rabin* [12], *Ateniese* and *de Medeiros* [2], *Chen* et al. [6] and *Zhang* et al. [20]. Additionally, a redaction based scheme proposed by *Miyazaki* et al. [16] was implemented to allow performance comparison between both approaches.

All schemes have been integrated into the Java Cryptography Architecture (JCA), to become usable for XML Signatures. We present XML Signature integrations of a representative subset of the different "flavours" of sanitizable signature schemes following the *Ateniese* scheme and redaction based schemes, which have not been developed explicitly for tree-structured documents. This integration is non-trivial: Sanitizable signature schemes do not work as straight forward as the standard hash-and-sign SHA–RSA-Scheme, since some schemes follow a different process when generating the signatures for document parts. In this paper we present our solution that integrates sanitizable signature schemes into XML Signatures without invalidating the W3C Standard.

Our contribution consists of the actual implementation and integration of five different schemes and an analysis that demonstrates a practical performance penalty, compared to RSA and SHA for most of the schemes presented. Additionally, we show how XML helps to mix and allows to add resp. remove properties from the original sanitizable signature scheme. We have shown this for the properties of transparency, accountability, consecutive sanitization control, restricting to values and restricting to sanitizers. These additions are still compliant with the XML Signature Specification and do not need major modifications of the underlying sanitizable signature algorithms. Finally, we offer two observations: First, not all sanitizable signature schemes are suitable for use in XML Documents, since some do not allow overlapping references. Second, the properties of transparency and its "flavours" weak and strong transparency, as originally defined by *Ateniese* et al. [1], are actually independent properties and should not imply each other.

The rest of the paper is structured as follows: Sec. 3 gives technical details on the integration into the existing frameworks and standards: JCA and XML Signature. A short summary of our detailed performance measures is given in Sec. 4. In Sec. 5 is shown how alterations using given primitives will add, respectively remove, certain properties from the schemes. We then use these alterations to motivate a revised view on the existing property of Transparency and its definition in Sec. 6.

2 Related Work

From an implementer's point of view, *Tan* et al. integrated the *Ateniese* scheme into XML Signature using one chameleon hash developed by *Krawczyk* et al. [12] in [18]. However, they just showed that one scheme can be integrated into XML

Signature without offering a performance analysis. Their implementation work is similar to that of *Ateniese* et al. in [1]: they also implemented the *Krawczyk* scheme, but in conjunction with OpenSSL and not JCA and XML Signature. Additionally, they did not discuss the possibility to alter properties of the schemes and their integration into the XML Signature Standard in more detail.

Brzuska et al. formalized the most common properties in [5], which we used for our work. However, they do not take XML Signature into account any further. They constructed a tag-based chameleon hash, which allows to add sanitizer accountability to the *Ateniese* scheme. We did not implemented this, as this chameleon hash is similar to the *Krawczyk* one.

Kundu et al. developed a sanitizable signature scheme, which addresses the specific needs of tree-structured documents [13]. These needs have been formalized by *Liu* et. al in [15]. This has also been addressed and tailored for redaction based signatures by *Bruzska* et al. in [4]. However, both approaches are not based on the hash-and-sign paradigm in the "natural way". For brevity their schemes are left out of this work. A performance analysis of the *Kundu* scheme can be found in [14].

Wu et al. also implemented a sanitizable signature scheme in [19]. However, they also do not provide a performance analysis and their scheme relies on the *Merkle*-Hash-Tree-Technique.

3 Implementation in JAVA and Integration into XML

We used the Java Cryptography Architecture (JCA) as an existing open-source framework and we implemented a new "Cryptographic Service Provider", following the design patterns given in JCA. Whenever the schemes required additional cryptographic algorithms these were added via external libraries. In particular: *GnuCrypto*[2] for EMSA-PSS [10], required by *Ateniese* [2], and a library from the *National University of Maynooth*[3] for elliptic curves and bilinear pairings, required by *Zhang* [20]. Our implementations make use of Java's `BigInteger` class.

Within the XML Signature Specification we use `<Reference>` to split the XML Document m into subdocuments $m_1, ..., m_n$. Subdocuments, or references, are actually pointers to subsets ($\subseteq$) of the whole document, e.g. a nodeset, an element or the complete document itself.[4] The union $m_1 \cup ... \cup m_n$ represents the part of the document which is covered by the final signature. Each of these references can be digested using a different hash-algorithm, allowing a maximum of flexibility for the signer. The resulting digests, along with additional information, are combined into the `SignedInfo` element. The additional information stored

[2] http://www.gnu.org/software/gnu-crypto/
[3] http://www.nuim.ie/
[4] *XPath* allows even more complex expressions.

contains, among others, the following: Applied digest method, C14n-algorithm, reference URI, and applied transforms. The `SignedInfo` element is canonicalized, hashed, and the resulting digest is signed. The W3C specification allows to assign parameters for each message-digest, we used this to integrate the keyed-hash-functions.

3.1 JCA Implementation Details for the Five Schemes

This section will shortly introduce the schemes we implemented and evaluated.

***Ateniese* Scheme.** Standard cryptographic hashes, denoted $\mathcal{H}$, do not allow attackers to find collisions in the hash-domain ("Collision-Resistance"). Chameleon hashes, denoted $\mathcal{CH}$, however do allow to compute collisions for arbitrary input, as long as a trapdoor is known; this trapdoor must be kept secret for chameleon signature schemes [12]. *Ateniese* et al. propose to use those hashes to construct a sanitizable signature scheme: [1]

$$\sigma = SIGN(d_1||...||d_n)$$

where

$$d_i = \begin{cases} \mathcal{H}(m_i) & \text{if } m_i \text{ is not sanitizable} \\ \mathcal{CH}(m_i) & \text{if } m_i \text{ is sanitizable} \end{cases}$$

$\mathcal{H}$ is a standard cryptographic hash like SHA-512, $\mathcal{CH}$ a chameleon hash, m_i subdocument i and n is the number of subdocuments.

Each sanitizer receives the trapdoor needed to find collisions, thus he is able to do arbitrary changes to the sanitizable subdocuments. For this work four different chameleon hashes have been implemented and used within the *Ateniese* scheme, as each chameleon hash can result in different properties of the resulting signature [5]:

1. *Krawczyk* as the first chameleon hash, based on the DLP assumption [12]
2. *Ateniese* as an ID-based approach [2]
3. *Zhang* as an ID-based approach without an UForge-algorithm [20]
4. *Chen* as an ID-based approach without the key-exposure-problem [6, 3]

As we are concerned with the application within XML, let us share the following observation concerning message splitting: For the *Ateniese* scheme it is necessary, that in general: $\forall i, j : m_i \neq m_j$, where $0 \leq i < j \leq n$. Suppose $m_i = m_j$ yields, which means that two subdocuments are the same. If now m_i is sanitizable while m_j is not, m_j cannot be altered since $SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m_j))$ must remain the same value under all circumstances. Since $\mathcal{H}$ serves as a random oracle, uniformerly distributing its digests over $dom(\mathcal{H})$, finding a collision such that $SIGN(\mathcal{H}(m_i')||\mathcal{CH}(m_j')) = SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m_j))$, is infeasible by definition of $\mathcal{H}$.

The prior statement is just correct if $m_i = m_j$. Let's assume that $m_i \subsetneq m_j$. Note, m_j can be expressed as $m_j = m_k||m_i||m_l$ where $k \neq i \neq l, m_k \neq \emptyset \vee m_l \neq \emptyset$. Further assume that m_i is not sanitizable while m_j is. Now, the sub-subdocument m_i which is contained within m_j cannot be altered while the complement $m_j \backslash m_i = \{m_k, m_l\}$ can. Let the signature be $\sigma = SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m_j))$ while $m_i \subsetneq m_j$. If now m_i is not changed in any way $\mathcal{H}(m_i)$ is not changed either. Since a collision in $dom(\mathcal{CH})$ can be found for arbitrary $m \in \{0,1\}^*$, m_i does not have to be changed. Therefore $m_j \setminus m_i = \{m_k, m_l\}$ can be altered in any way, since $SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m_j)) = SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m'_j)) \wedge SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m_j)) = SIGN(\mathcal{H}(m_i)||\mathcal{CH}(m'_k||m_i||m'_l))$ remains true. This is important if the document to be sanitized follows a strict ordering, i.e. text-documents. Note, for $m_j \subsetneq m_i$ this does not work, since all alterations to m_j also affect m_i, which means that the resulting digest $d_i = \mathcal{H}(m_i)$ will differ. One may argue that splitting the document up into three parts, namely m_k, m_i and m_l, while making m_k and m_l sanitizable fulfills the same requirements. This is possible if the subdocument itself is just text. If additional information is signed, i.e. structure, or the node-based nature of XML documents is taken into account this may not be always possible. Consider an *XPath* expression that counts the number of elements and is signed using a standard cryptographic hash, while the nodeset used by this expression is signed using a chameleon hash. This prohibits the alteration of the number of elements contained. Since *XPath* is *Turing*-Complete [8] this construction allows the protection of all structural information, if the expression used is wisely chosen. A formal construction of these expressions is future work.

***Miyazaki* scheme** *Miyazaki* et al. developed a redactable signature scheme

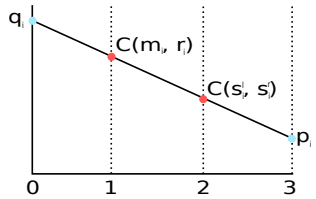

Fig. 1. Illustration of the *Miyazaki* Scheme

which allows disclosure control in [16]. The scheme just allows deletion and offers the possibility to prohibit additional sanitization for a consecutive sanitizer. Each subdocument m_i is transfered on a coordinate system. The transform is a commitment to m_i using e.g. the *Halevi*-Commitment-Scheme [9]. This commitment $(C(m_i, r_i))$[5] will then be transfered onto the point $(1, C(m_i, r_i))$. In a second step, two random numbers s_i^l, s_i^r with $|s_i^l| = |s_i^r| = |r_i|$ are committed to by calculating an additional commitment $C(s_i^l, s_i^r)$. The second commitment constructs the point $(2, C(s_i^l, s_i^r))$. Finally, the signer calculates two auxiliary

[5] Note, the *Halevi* scheme requires a purely random number r

points $(3, P_i)$ and $(0, Q_i)$, as illustrated in Fig. 1. The resulting signature is $\sigma_{miyazaki} = (P_1||...||P_n||Q_1||...||Q_n)$. The signature $\sigma_{miyazaki}$, all P_i, all m_i, and the corresponding (de)commitment values will be distributed with m. To sanitize a sanitizer just has to remove m_i and r_i along with its (de)commitment values; a verifier is able to reconstruct Q_i given P_i and $C(s_i^l, s_i^r)$. To prohibit sanitization a sanitizer has to remove s_i^l and s_i^r along with the (de)commitment values; a verifier is still able to verify the signature, calculating Q_i using $C(m_i, r_i)$ and P_i. Removing both values will result in a non-verifiable signature. *Miyazaki* requires completely disjunct subdocuments ($\forall i, j : m_i \cap m_j = \emptyset$ where $0 \leq i < j \leq n$)[6], since the scheme just allows deleting subdocuments rather then arbitrary alterations. Hence, any partial deletion will alter another subdocument which is not possible with this scheme. This limits its usefulness in XML Signatures, since *Miyazaki* does not allow overlapping references.

3.2 Integration into an XML Signature

In this section we give a brief introduction on the JCA code changes resp. extension made to allow XML Signatures with the sanitizable schemes. Sources will be made available upon request. However, we present all the modifications necessary to make the schemes JCA implementable.

Key Generation. Every chameleon hash needs a key-pair. Obviously this key-pair has to be generated prior to calculating the digest. A Java key-generator accepts exactly two parameters, a `SecureRandom` object, which handles the generation of random numbers and a security parameter τ used to define the key-length. However, the class `BigInteger` does not allow to generate primes which have a "bit-range". Hence, each key will be generated using $\tau_{new} = \tau - 1$ to avoid too huge primes. This reduces the security by two Bits. Trivially, the workaround is to increase the bitlength by two to get at least the same security level. For the performance evaluation this has no major impact.

Key Generation Zhang. For the *Zhang* chameleon hash no keys have been defined, since this scheme has been implemented as a TTP-Service.

DigestMethod. The chameleon hashes need a public-key to be computed. We marshalled the public key into to the reference element. For example, public keys in the *Krawczyk* scheme have four elements: $PK_{CH} = (p, q, g, y)$ [12]. We add a new element containing the public key as *Base64*-encoded `BigInteger`s as illustrated in Lst. 1.1. Finally, we used new URIs to identify each digest method[7].

For the *Zhang* scheme the coin consists of a point P on an elliptic curve $E(\mathbb{F}_q)$. The point is stored as its (x, y) coordinates.

For the *Miyazaki* and *Chen* scheme some workarounds were necessary due to the JCA not being able to store negative numbers. Therefore, everything must

[6] They need to form a disjunct partition

[7] `http://www.example.org/xmldsig-more#chamhashdisc`, `...#miyazaki`, etc.

Listing 1.1. Marshalling of the *Krawczyk* Parameters

```
<DigestMethod Algorithm="http://www.example.org/xmldsig-more#chamhashdisc">
  <ChamHashDiscKeyValue>
    <p>Aa5Mue7ppx2YD7R8KXUqQIKSTSay6jHhWm9L0dxHpL2P</p>
    <q>1yZc93TTjswH2j4UupUgQUkmk111GPCtN6Xo7iPSXsc=</q>
    <r>FQrJPkWb0JwiffjrAdbWAoyropQmNohMgEy6ABsvptQ=</r>
    <g>JtqJ1H0NL0Is+6Y797XKQ1hbHc+HYgoGQAkvK8h+q8Y=</g>
    <y>AVwdxMlXF6HIHRHl0r7Xoojb0VoB7ZBP4Dxc83BDDgxG</y>
  </ChamHashDiscKeyValue>
</DigestMethod>
```

exclusively be encoded using positive numbers. For the *Chen* scheme we store a negative b, i.e. -1, as a Base64-encoded 2 and reverse it to -1 during unmarshalling. For the *Miyazaki* scheme a negative point P is not unlikely, if $\Delta = C(s_i^l, s_i^r) - C(m_i, r_i) < 0$. To prohibit this the result of $C(s_i^l, s_i^r)$ is multiplied by $2^{|p|-1}$. This ensures that $\forall C(m_i, r_i) : \Delta > 0 \Rightarrow P > 0$, where p is the prime used in the *Halevi* scheme. Note, neither of the workarounds impacts security since no Bits are omitted and the lower Bits are shifted out during unmarshalling.

Our second workaround in the implementation of the *Miyazaki* scheme addresses the output of the reference digest, which is $(P_i||Q_i)$ for each reference, since the references are sequentially processed by the JCA framework. Hence, we add an additional hashing step which extends the reference "digest" to $\mathcal{H}(P_i||Q_i)$ in order to gain a fixed length output. Additionally, any deletable element has been placed in an `Signature` element not covered by the signature itself. Trivially, this must be done to ensure that the signature verification can be done correctly if any changes are done. E.g. for the *Miyazaki* scheme the (de-)commitment values are added outside of `SignedInfo` as shown in Lst. 1.2, since they can be removed by a sanitizer. Note, we link them to the corresponding references by using the reference's `URI` value as `Id` attribute.

For the chameleon hashes the random value r resp. b are also part of the `DigestMethod` element, as shown in Lst. 1.1. Just to be clear: This removes the transparency property. However, storing them here or forward referencing them has negligible impact on the performance itself. Another way to allow random coin change would be to add an additional transform which removes them prior to digesting the `Reference` element. The latter approach was not chosen since an additional transform would not have been as self-explaining as this version.

Additional Hashing Steps. The hashes proposed by *Ateniese* et al., the one proposed by *Zhang* et al., and *Chen* et al. perform an additional standard cryptographic hash prior to its own calculations. To have comparable evaluations this hashing-step has also been added to the other schemes. This additional hashing step has been fixed as SHA-1 for all schemes to have comparable results. The commitment-scheme required by the *Miyazaki* scheme returns an array (y_i, a_i, b_i, p_i). For further calculations y_i, a_i, b_i have to be merged into one in-

Listing 1.2. Marshalled *Miyazaki* scheme (de-)commitment values

```
<Signature>
  <SignedInfo>...</SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>...</KeyInfo>
  <MiyazakiKeyValue Id="#xpointer(id('8492341'))">
    <sr>cjyt7T7qu3j+ieBksyEOJ+yVv/0=</sr>
    <sl>S5NrovXVK0YkswUERiz0EfjR+fc=</sl>
    <r>bWTFplIzjCpiCWReRGzCL8m2OyY=</r>
    <a>DKFCyGUL</a>
    <b>ASWQM3JEo0cpm77v0oc+NLwujCX+</b>
    <y>7Zo2Etyvy3Umwf8LlyRfDSCvLc4=</y>
    <a_s>CbaVZ/t+Lw==</a_s>
    <b_s>AbW5KeKOBcyQMxqpZ0oqLUIwakDU</b_s>
    <y_s>W9k9TudAZ8l3C4gFXbA1/VM11E8=</y_s>
    <prime>AuX1hF/7HhBr5va4Ayx9HWIKlSfj</prime>
    <prime_s>Ao8RKY6Ezd5uRikVEpLheUqxdYVz</prime_s>
  </MiyazakiKeyValue>
</Signature>
```

Listing 1.3. Appended Values for the *Chen* scheme

```
<Signature>
  <SignedInfo>...</SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>...</KeyInfo>
  <r Id="#xpointer(id('8492340'))">210874650874</r>
  <b Id="#xpointer(id('8492340'))">2</b>
</Signature>
```

teger. We pad all numbers with zeroes till they have the same size. This means that $|q_{i_1}|+|y_i| = |q_{i_2}|+|a_i| = |q_{i_3}|+|b_i|$, where $q_i \geq 0$ are the padding Bits. This does not allow any modifications since the width of each part is fixed. Note, the prime p is an external value and thus not part of the commitment-value itself.

Signing. For the *Miyazaki* scheme the digest values need to be added prior to hashing the reference. Thus, all parameters added to the `DigestMethod` element must be known prior to the digestion procedure. This would have prohibited the calculation of commitments. Our workaround runs the signing process twice, in the first run the intermediate results, i.e. the (de-)commitment values are cached and added during the second run. The second run's results will not be used. A negative performance impact, but this approach did not require major code changes in the JCA framework itself.

Sanitization. The sanitization procedure outputs an r'_i (and b'_i for the *Chen* scheme) for a given m'_i such that $\mathcal{CH}(r_i, m_i) = \mathcal{CH}(r'_i, m'_i)$ where $m_i \neq m'_i$ and $r_i \neq r'_i$. For performance reasons we cache each subdocument during the signing process to allow an easy UForge implementation. Lst. 1.3 shows how a sanitizer adds new elements to the `Signature` element for the *Chen* scheme, which requires the two values b'_i and r'_i. For *Ateniese* and *Krawczyk* the element

contains only r'_i for each sanitized reference. The *Zhang* scheme requires a point (x'_i, y'_i).

The values leading to a collision are not encoded in *Base64*; this was done for convenience. A sanitizer can add the calculated values without additional encoding, except the encoding for negative values. Note, this does not have a security-impact; a maliciously changed r'_i will most likely result in a wrong digest value. Thus, the whole signature validation will fail and malicious alterations will be detected. In our performance evaluation we replaced every sanitized subdocument by the fixed string `<test id="ID">xxx</test>`.

For the *Miyazaki* scheme we have a different sanitization process: The whole subdocument has to be replaced with a new element `<sanitized id="ID> ARBITRARY STRING </sanitized>`. This allows for an easy decision which commitment value to use when calculating the auxiliary points. Additionally, the corresponding de-commitment values have to be removed as well.

Special Implementation of the *Zhang* scheme. The chameleon hash proposed by *Zhang* et al. requires a very large signature and the reconstruction of $E(\mathbb{F}_q)$ for each validation. We used a client-server approach. Every method of the chameleon hash is calculated by a TTP and transfered to the calling application. This was done to avoid reconstructing the elliptic curve for every validation and to have a smaller signature, which just contains the random point $(x_i, y_i) \in E(\mathbb{F}_q)$ and the recipient S for each reference. Note, this use of an online-TTP is not enforced by the scheme itself; the verification only needs the public key to reconstruct the curve.

4 Performance Evaluation of Implemented Schemes

For each performance measurement we performed 100 runs and calculated the median to reduce the impact of the probabilistic algorithms. Every input is digested using a standard cryptographic hash prior to hashing it with the chameleon ones. Thus, the results are independent of subdocument's length. For every identity-based scheme we used a fixed S as well.

Tests were run on a *Lenovo Thinkpad T61* with an *Intel* T8300 Dual Core @ `2.40` Ghz and `4` GiB of RAM. The operating system was *Ubuntu* Version 10.04 (64 Bit), while the Java-Framework version was `1.6.0_20-b02`.

4.1 Algorithms: Setup, Hashing and Forging

To have a similar and commonly known algorithm to compare with, the RSA key-pair-generation-algorithm was also measured. e has been fixed to 65537 in the RSA algorithm. The results are shown in Tab. 1 along with SHA-512 for hashing.

Algorithm	Setup	Hash	IForge	UForge
RSA/SHA-512	400,119	7	-	-
Krawczyk	11,082,481	3	4	16
Ateniese	7,856	410	424	522
Chen	68,319	1,878	248,280	7,661
Zhang 512Bit-Key	9,570,008	25,547,333	8,267,775	-
Zhang 128Bit-Key	243,040	929,648	381,215	-
Miyazaki [a]	4,700	generation: 11,400 verification: 70	0	-

Table 1. Median Runtime: Input: 160 Bit; Hash-Output: 512 Bit; all in μs

[a] Miyazaki does not use hashing, times are given for the comparable operation.

Krawczyk et al. The key pair generation time was the longest. It can basically be divided into the need to find a safe-prime $p = 2q + 1$ and a generator of order q, generating $\mathbb{Z}/p\mathbb{Z}$. Detailed analysis found the generation of the safe-primes consumes the most time (99%), while finding a generator and setting up the keys is negligible with less than 1% of the whole key generation time. Our comparison yielded that key-length has the biggest performance impact for *Krawczyk*, especially notable during key-pair generation. Currently the highest practical key-size is 512 Bit. Hashing and forging do not use much time and are almost calculated instantly. We suggest a pool of pre-generated key-pairs to avoid long wait times for new key-pairs.

Ateniese et al. The key-pair-generation of the *Ateniese* scheme is not as expensive as the *Krawczyk* scheme since no safe-primes are needed. We found the prime generation to be the limiting factor; all other operations are almost instantly calculated. The argumentation is basically the same as for the *Krawczyk* scheme. The setup is fast and even huger security parameters can be used. However, hashing and forging are more complex algorithms than *Krawczyk*'s and take longer. Pre-generated primes could increase the performance. Compared to *Krawczyk*'s it will be faster if you consider that *Krawczyk* would require new keys, while the ID-based concept used by *Ateniese* just needs one "master-key-pair". In practice, the speed gained during key generation compensates the longer hashing times.

Chen et al. As the *Ateniese* scheme, the *Chen* scheme does not require safe-primes. However, the needed primes cannot be arbitrary, since $p \equiv q \equiv 3 \pmod 4$ must be true. ω has been fixed to 64 for all measurements. Notably, *Chen*'s *IForge* algorithm takes a lot of time. Further investigation showed the extraction process to consume the most time, in particular the square root calculation is very expensive, consuming $> 99\%$ of the time. However, the practical impact is reduced since *Chen* is key-exposure-free. Compared to *Ateniese*, which does not offer this property, the impact of key-exposure must not be limited. *Ateniese* et al. suggested in [2] to append a `TransactionID` to the `ID`, resulting in the extraction procedure being performed more often in *Ateniese* than in

Chen. A practical performance increase for the TTP for *Chen* is to extract the trapdoor along with the registration or on request, as *Ateniese* proposed for his scheme [2].

***Zhang* et al.** The *Zhang* scheme relies on elliptic curves and pairings. Hence, the Key-Pair-Generation is actually constructing an elliptic curve along with its parameters. Note, the evaluation of the *Zhang* scheme includes the delays caused by our TCP-socket-connections. Additionally a delay of $1s$ had to be introduced to avoid a timeout of the signature-generation due to the TCP-transfers and calculation on the server-side. This implementation-specific overhead has been subtracted from the presented timings. Our observation showed that even after the curve has been constructed, the setup for *Zhang* still takes a long time. This is caused by adding the system parameters; however due to the library approach no further investigation was possible.

The *Zhang* scheme suffers from certain limitations; first of all the key-pair-generation is very slow. Tab. 1 shows that for a key length of 512 Bit the *Zheng* algorithms take considerately more time. The speeds are getting practical when the key length for the elliptic curve based scheme is lowered to 128 Bit, which offers comparable security. The practical impact of the slow *IForge* algorithm depends on how often sanitization is done.

Miyazaki et al. Finally, the *Miyazaki* scheme. It works completely different; instead of calculating a digest it calculates a commitment, which can be seen as a key-pair. Obviously, most time is spent calculating the commitments $(11,400\mu s)$, while verifying a commitment is almost instantly $(70\mu s)$. As for the other schemes, this is due to the fact that primes must be generated. Redaction, comparable to *IForge*, is just removing the element from the XML, and does not require any calculation at all.

4.2 Signature Generation and Verification

Comparing algorithms with such different approaches is like comparing apples and oranges; however, all schemes aim to provide sanitizable signatures. The preceding section presented the bare runtime of each algorithm. To see how the different schemes scale in praxis, they have been used to generate an XML signature over the document listed in Lst. 1.4. By `xpointer`s we split the document into two subdocuments; namely `#xpointer(id('8492340'))` and `#xpointer(id('8492341'))` have been used, returning the subtrees `Item` and `Address`.

For this evaluation each scheme will be measured with 512 Bit keys, while the key-generations, both for the underlying RSA-Signature and the chameleon hashes, have been omitted to have comparable results. This was done since the *Zhang* scheme relies on a TTP to generate the curve on start-up. SHA-512 has been measured as well to see the performance impact of a sanitizable schemes. Tab. 2 shows the results for a "real" signing and verification step. As before, to

Listing 1.4. The XML File used

```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder>
 <Item id="8492341">
  <Description>Video Game</Description>
  <Price>10.29</Price>
 </Item>
 <Buyer>
  <Name>My Name</Name>
  <Address id="8492340">
   <Street>One Network Drive</Street>
   <Town>Burlington</Town>
   <State>MA</State>
   <Country>United States</Country>
   <PostalCode>01803</PostalCode>
  </Address>
 </Buyer>
</PurchaseOrder>
```

	SHA-512	*Krawczyk*	*Ateniese*	*Chen*	*Zhang 128*	*Zhang 512*[a]	*Miyazaki*
Generation	2,010,624	2,219,806	2,472,972	2,424,781	3,645,070	52,434,635	2,675,284
Validation	1,373,907	1,301,410	1,366,284	1,278,295	1,908,047	50,256,292	1,339,633

Table 2. Scheme Comparison (Generation/Validation) for 512 Bit keys in μs

[a] Zhang with 128 Bit offers a comparable security.

have meaningful results a 1s delay was subtracted for the *Zhang* scheme and the server was already running.

4.3 Summary

Surprisingly, our evaluation showed that all schemes come with similar runtime figures for generation and for validation; this finding, however, does not account for the pre-generated key-pair. Taken the key-pair generation into account as well, the chameleon hash proposed by *Ateniese* et al. achieves the best performance, in particular since just one key-pair has to be generated. Note, that it suffers from the key-exposure-problem as discussed in [2]. Using transaction-based IDs helps reducing the key-exposure-problem's impact, hence the *Ateniese* scheme is not as heavily affected as the *Krawczyk* scheme. *Krawczyk* needs a new key-pair for each subdocument to prohibit non-wanted key-exposure. Thus, all ID-based chameleon hashes perform better in this respect, even if they require more complex algorithms. The notable exception is the *Zhang* scheme, which requires calculations on elliptic curves. However, cryptographic algorithms based on ECC offer a higher degree of security. Hence, smaller key-sizes are acceptable which decreases the time needed for the algorithms.

The *Miyazaki* scheme is not as suitable for XML files as the *Ateniese* scheme regardless of the chameleon hash. Since in *Miyazaki* it is not possible to protect

structural information as it requires non-overlapping subdocuments. However, we showed it has a comparable runtime.

5 Property Changes of Schemes due to Mixing

The schemes introduced suffer from certain limitations, e.g. a verifier is able to figure out what parts are sanitizable in the *Ateniese* scheme by just looking at the used digest method. We propose some additions to these schemes which add resp. remove certain properties.

5.1 Properties of Sanitizable Signature Schemes

Brzuska et al. already formalized most of the properties given in [5]. Hence just the basic idea is given here.

1. *Unforgeability:* For an outsider (i.e. for anyone not being the signer or a sanitizer) it is infeasible to forge signatures.
2. *Immutability:* The sanitizer should not be able to change parts of the document which are not designated to be sanitized.
3. *Privacy:* Sanitized parts should not leak any information which may be used to reconstruct information which has been censored.
4. *Transparency:* It should be impossible for the verifier to decide whether a given document was sanitized or not. *Ateniese* et al. in [1] further distinguish between:
   - *Weak Transparency:* Weak Transparency means that the verifier is able to identify the subdocuments m_i that can potentially be sanitized.
   - *Strong Transparency:* Is the opposite of Weak Transparency; the recipient is not able to distinguish whether a subdocument m_i is sanitizable or not.

   To avoid confusion, *Brzuska* et al. formalized only *Ateniese* et al.'s notion of Weak Transparency under the term Transparency [5].
5. *Accountability:* The responsibility of a message should not be transferable to another party, in particular the original signer should not be held responsible for a maliciously sanitized message. *Brzuska* et. al showed that this property must be split up as well: [5]
   - *Signer Accountability:* If a document was not signed by the signer even a sanitizer should not be able to accuse him.
   - *Sanitizer Accountability:* If a document was signed by a signer, the signer should not be able to accuse the sanitizer if he did not alter document.

Not formalized by *Bruzska* et al. have been the following properties:

6. *Restrict to Values:* A sanitizer is just able to replace a subdocument with certain preset values instead of ones of his own choice.
7. *Sanitization Control:* Defines if a signer has control over the parts which can be sanitized by a sanitizer.
8. *Consecutive Sanitization:* Defines if an already sanitized document can be sanitized again by another sanitizer.
9. *Consecutive Sanitization Control:* Defines if a sanitizer can prohibit further sanitization by another sanitizer.

5.2 Extensions of the *Ateniese* Scheme

This subsection will introduce some additions to the *Ateniese* scheme, using a mixture of given primitives. The proposed extensions will lead to the redefinition of transparency in Sec. 6.

Removing Transparency The *Ateniese* scheme [1] is defined as $\sigma_{ateniese} = SIGN(d_1||...||d_n)$, where

$$d_i = \begin{cases} \mathcal{H}(m_i) & \text{if } m_i \text{ is not sanitizable} \\ \mathcal{CH}(m_i) & \text{if } m_i \text{ is sanitizable} \end{cases}$$

This allows to see which subdocuments m_i are potentially sanitizable, but not if they have actually been sanitized. We change this by adding the original r_i for each chameleon hash to allow a "m_i has been sanitized" detection. Therefore the signature is expanded to $\sigma'_{ateniese} = SIGN(s_1||...||s_n||d_1||...||d_n)$, where

$$s_i = \begin{cases} \text{The } r_i \text{ in } \mathcal{CH}(m_i, r_i) & \text{if } m_i \text{ is sanitizable} \\ \emptyset & \text{else} \end{cases}$$

Every s_i is covered the signature in cleartext. This implies that every recipient is able to decide whether a given (sub-)document has been sanitized, since he can compare the r'_i used for calculating the chameleon hashes with the s_i provided by the signature $\sigma'_{ateniese}$. This has no impact on security, because the verifier is still not able to find additional collisions since he needs the original m_i. This is the scheme implemented in this paper, as proposed by *Tan* et al. [18].

Removing Transparency II The previous approach can be fine-tuned further. The r_i used to calculate each chameleon hash are now not directly appended to the signature, but hashed using a cryptographic hash, i.e.: $\sigma''_{ateniese} = SIGN(d_1||...||d_n||h_1||...|h_n)$ while $h_i = \mathcal{H}(s_i)$. However, this construction can be enriched with an interesting flavour; the hash $\mathcal{H}$ could also be replaced with a chameleon hash ($h_i = \mathcal{CH}_{trans}(s_i)$), therefore allowing only the sanitizer who holds an additional private key sk_{trans} to transparently sanitize a subdocument.

This extended scheme requires that every r_i is part of the respective hash; however it is possible that the property of non-transparency is just desired for certain subdocuments. Obviously, this can be achieved by removing r_i from the concatenation. This allows a signer to decide where he wants that property. However, an attacker cannot add r_i' to falsely indicate a sanitization; the verifier can use the r_i for verification which will reassure him that m_i was not sanitized. The trivial proof is omitted for brevity.

Adding Strong Transparency *Ateniese* et al. already note in [1] that it is very simple to gain strong transparency by generating distinct key pairs for each chameleon hash, i.e.

$$d_i = \begin{cases} \mathcal{CH}_{k_i}(m_i) & \text{if } m_i \text{ is sanitizable} \\ \mathcal{CH}_{u_i}(m_i) & \text{if } m_i \text{ is not sanitizable} \end{cases}$$

We will later make use of this approach. This can also be used to improve the overall performance. Originally, each chameleon hash needed a separate key pair; However, if the standard chameleon hash is replaced with an ID-based construction, just S needs to be different for each generated hash since the private key B differs. This leads to another interesting application, a signer can define groups, similar to networking domains. This means the TTP distributes the secret keys with respect to a sanitizer's group membership.

A similar approach without using a TTP is to use the extended chameleon hash presented by *Ren* et al. which allows a multi-user hash [17]. Hence, a signer can select which sanitizers are able to sanitize while hiding the group membership.

Adding Restricting to Values One way to restrict someone to values is the use of an chameleon hash without an *UForge* algorithm, like the one proposed by *Zhang* et al. [20]. An additional way is to use Bloom-Filters as proposed by *Klonoswki* et al. [11].

5.3 *Miyazaki* Scheme

This section will introduce some mechanisms which change the properties of the *Miyazaki* scheme.

Adding Weak Transparency Adding weak transparency can be achieved by adding a random number of "fake-documents" between the real subdocuments. This fake-subdocuments will be redacted (sanitized) by the signer itself prior to sending it to the sanitizers. As a result the signer gives only sanitized documents to the first sanitizer. Thus, neither a sanitizer nor a verifier knows if the document has been sanitized by sanitizers, since he cannot distinguish between a sanitized original subdocument and a sanitized fake-subdocument. However, the signer now no longer distributes technically not sanitized documents, which might not be wanted in all use cases.

Adding Accountability, Consecutive Sanitization Control & Restricting to Sanitizers A way to add both signer and sanitizer accountability is to use a similar mechanism as with the *Ateniese* scheme. The signer adds an additional chameleon hash over the whole message m to the signature using a tag-based-chameleon hash, i.e. the one proposed by *Brzuska* et al. in [5].
$\sigma_{miyazaki} = SIGN(d_{ch}||P_1||...||P_n||Q_1||...||Q_n)$ where $d_{ch} = \mathcal{CH}_{tag}(m)$.
If a subdocument is redacted now, the tag-based-chameleon hash needs to be adjusted as well. This allows a signer to trace back the changes and thus to find a malicious sanitizer. Note, a sanitizer is not able to do arbitrary changes to the subdocument since the signature still covers the commitments. This construction also implies that just sanitizers defined by the signer are able to alter a signed document, since only they know the trapdoor needed to calculate collisions.

To prohibit consecutive sanitization control an additional chameleon hash over the (de-)commitment values of the mask can be constructed. The argumentation is essentially the same as before.

6 The Transparency Property Revisited

Ateniese et al. define the property of transparency (T) as follows:

> Given a signed message with a valid signature, no party — except the censor and the signer — should be able to correctly guess whether the message has been sanitized [1].

They further divide the property into "weak" (WT) and "strong transparency" (ST):

> We further distinguish among two flavors of transparency: weak and strong. Weak transparency means that the verifier knows exactly which parts of the message are potentially sanitizable and, consequently, which parts are immutable. In contrast, strong transparency guarantees that the verifier does not know which parts of the message are immutable and thus does not know which parts of a signed message could potentially be sanitizable [1].

Essentially, T always implies exactly one of WT or ST and vice versa.

$$(T \implies (ST \dot{\vee} WT)) \wedge (ST \dot{\vee} WT)) \implies T) \equiv T \Leftrightarrow (ST \dot{\vee} WT)$$

where $\dot{\vee}$ denotes "exclusive or". Practically, a verifier either knows which m_i is potentially sanitizable or he does not. Hence, $(ST \dot{\vee} WT)$ should be a tautology, i.e. $\models (ST \dot{\vee} WT)$. Since $T \Leftrightarrow (ST \dot{\vee} WT)$ this results that T is always true. This is counterintuitive and contradicts the *Miyazaki* scheme. Also, our construction in Sec 5.2 removes transparency in general on the message level, but leaves WT defined on subdocument level untouched.

Strong transparency does not always imply transparency either, as the following construction demonstrates: Let $\sigma = SIGN(d_1||...||d_n||\mathcal{H}(m))$ where all d_i are calculated as in the *Ateniese* scheme with strong transparency. (Sec. 5.2) Now a verifier is given the digest $\mathcal{H}(m)$ along with message m' and the signature σ. Note, we make the hash of the original available. This puts a special requirement on the signed message entropy: It needs to provide enough entropy to resist an attack that reconstructs the original message from the hash and the sanitized message. However, our solutions impacts on privacy, as defined by Brzuska et al. [4], since a single bit is leaked. A simple solution to avoid the possibility to reconstruct a message would be to add random numbers μ_i to each subdocument m_i, i.e. $m_{i_{new}} \leftarrow m_i||\mu_i$. Note: This is just usable for messages $|m| \ggg 1$. Let m' be the potentially sanitized message. The verifier uses m' and $\mathcal{H}(m)$ to verify the signature σ over m'. With this construction a verifier can deduce that m' is a sanitization, hence that it has been altered by comparing $\mathcal{H}(m)$ and $\mathcal{H}(m')$, but gains no knowledge about what parts m_i are potentially sanitizable. Hence, this scheme does not have the property of transparency while it maintains the property of strong transparency. On a closer look, the scheme given in 5.2 can also be modified to fulfill this requirement; the only addition is to hash the *concatenation* of $r_1, ..., r_n$ prior to signing, i.e. $\sigma = SIGN(\mathcal{H}(r_1||...||r_n)||d_1||...||d_n)$. The verifier is provided with $\mathcal{H}(r_1||...||r_n)$. He further knows each chameleon hash's actual value r'_i, which can be different or equal to r_i. If the verifier compares $\mathcal{H}(r_1||...||r_n)$ from the signature with $\mathcal{H}(r'_1||...||r'_n)$ he can identify that m was sanitized, but cannot deduce which m_i.

A scheme without transparency but with weak transparency can be constructed in a similar way; instead of using the *Ateniese* scheme with strong transparency the standard one is used. This allows to see what subdocuments are potentially sanitizable but only to deduce if the document as a whole has been sanitized.

We conclude that the properties of weak and strong transparency are actually independent from the property of transparency. Hence, we do not see them as "different flavours" [1] as stated by *Ateniese* et al. However, *Ateniese* et al. correctly differentiated a difference in scope linguistically. We want to emphasize this further by explicitly differentiating the scope:

> Transparency makes a statement about a sanitiz***ed*** document as a whole.
> Weak and Strong Transparency make statements about sanitiz***able*** ***sub***documents.

7 Conclusion

We have integrated sanitizable signature schemes into the existing XML Digital Signature Specification without any major adjustments, and by disregarding the workarounds caused by bugs in the JCA. This also covers schemes which do not rely on the standard hash-and-sign paradigm like the *Miyazaki* scheme and schemes based on a TTP. Moreover, we showed that the performance of all

schemes are comparable to the standard SHA-512 - RSA signature procedure. Furthermore, we have shown that the existing schemes are extendable while still being compliant with the XML Digital Signature Specification. This also lead to the finding that the properties of Weak and Strong Transparency, covering subdocuments, is independent from the overall property of Transparency, which concerns the document as a whole.

8 Acknowledgements

The authors like to thank Gene Tsudik and Christina Brzuska for their comments and the fruitful discussions we had after presenting the original paper at ACNS 2011.

References

1. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable Signatures. In *ESORICS*, pages 159–177, 2005.
2. Giuseppe Ateniese and Breno de Medeiros. Identity-Based Chameleon Hash and Applications. In *Financial Cryptography*, pages 164–180, 2004.
3. Giuseppe Ateniese and Breno de Medeiros. On the Key Exposure Problem in Chameleon Hashes. In *4th International Conference on Security in Communication Networks*, LNCS 3352, pages 165–179. Springer, 2004.
4. C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *Proceedings of Applied Cryptography and Network Security*, pages 87–104. Springer, 2010.
5. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In *Proceedings of PKC*, pages 317–336. Springer-Verlag, 2009.
6. X. Chen, H. Tian, and F. Zhang. Comments and Improvements on Chameleon Hashing Without Key Exposure Based on Factoring. In *IACR Cryptology ePrint Archive*, number 319, 2009.
7. Eastlake, Reagle, and Solo. XML-signature syntax and processing. W3C recommendation. `www.w3.org/TR/xmldsig-core/`, Feb. 2002.
8. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *Proceedings of the 22nd Symposium on Principles of Database Systems*, PODS, pages 179–190, New York, USA, 2003. ACM.
9. Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In *CRYPTO '96*, pages 201–215. Springer, 1996.
10. J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), 2003.
11. Marek Klonowski and Anna Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.
12. Hugo Krawczyk and Tal Rabin. Chameleon Hashing and Signatures. In *Symposium on Network and Distributed Systems Security*, pages 143–154, 2000.

13. A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand, 2008. ACM.
14. A. Kundu and E. Bertino. CERIAS Tech Report 2009-1 Leakage-Free Integrity Assurance for Tree Data Structures, 2009.
15. Baolong Liu, Joan Lu, and Jim Yip. XML Data Integrity Based on Concatenated Hash Function. *CoRR*, abs/0906.3772, 2009.
16. Kunihiko Miyazaki, Mitsuru Iwamura, and et al. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 2005.
17. Qiong Ren, Yi Mu, and Willy Susilo. Mitigating Phishing by a New ID-based Chameleon Hash without Key Exposure. In *Proceedings of AusCERT*, 2007.
18. Kar Way Tan and Robert H. Deng. Applying Sanitizable Signature to Web-Service-Enabled Business Processes: Going Beyond Integrity Protection. *Web Services, IEEE International Conference on*, 0:67–74, 2009.
19. Zhen-Yu Wu, Chih-Wen Hsueh, Cheng-Yu Tsai, Feipei Lai, Hung-Chang Lee, and Yufang Chung. Redactable Signatures for Signed CDA Documents. *Journal of Medical Systems*, pages 1–14, December 2010.
20. Fangguo Zhang, Reihaneh Safavi-naini, and Willy Susilo. ID-Based Chameleon Hashes from Bilinear Pairings. In *IACR Cryptology ePrint Archive*, number 208, 2003.