UNIVERSITÄT PASSAU

ISL Institute of IT-Security and Security Law

Sanitizable Signatures in XML Signature

Henrich C. Pöhls, Kai Samelin, Joachim Posegga

Chair of IT Security, University of Passau, Germany

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|jp)@sec.uni-passau.de, samelin@fim.uni-passau.de

A Sanitizable Signature Scheme allows

- a defined third party („sanitizer")
- to alter defined parts of an already signed document
- without invalidating the given signature,
- without interaction between signer and sanitizer.

Sanitizable Signature Scheme

Generally consists of five algorithms:

1. **Setup**: Generate key pair and public parameters
2. **Sign**: Generate a Signature over sanitizable and immutable parts
3. **I-Forge**: Change sanitizable parts such that signature is still valid. Requires knowledge of the „sanitizer secret“.
4. **U-Forge**: Change sanitizable parts such that signature is still valid. Requires two different „versions“, i.e. original and sanitized msg.
5. **Verify**: Verify signature‘s validity. Valid iff immutable parts are unchanged and changing the sanitzed parts using I-Forge or U-Forge.

Contribution

- **Our Implementation of 5 Schemes in JAVA yields:**
  - Tolerable Performance penalty compared to SHA/RSA (most of the schemes)
  - Integration into JAVA Crypto Framework (JCA) possible (as a new JAVA Crypto Provider)
  - Integration into XML Digital Signature Syntax and Processing Standard (W3C) possible
- **Changing Properties by Mixing Chameleon- and SHA-Hashes**
- **More Precise Definition of 3 Properties Transparency and Strong- / Weak-Transparency**

Research Context

ReSCUe IT

Project Goal: IT-supported Robust & Secure Supply Chains

Goal of University of Passau / Institute of IT Security and Security Law:

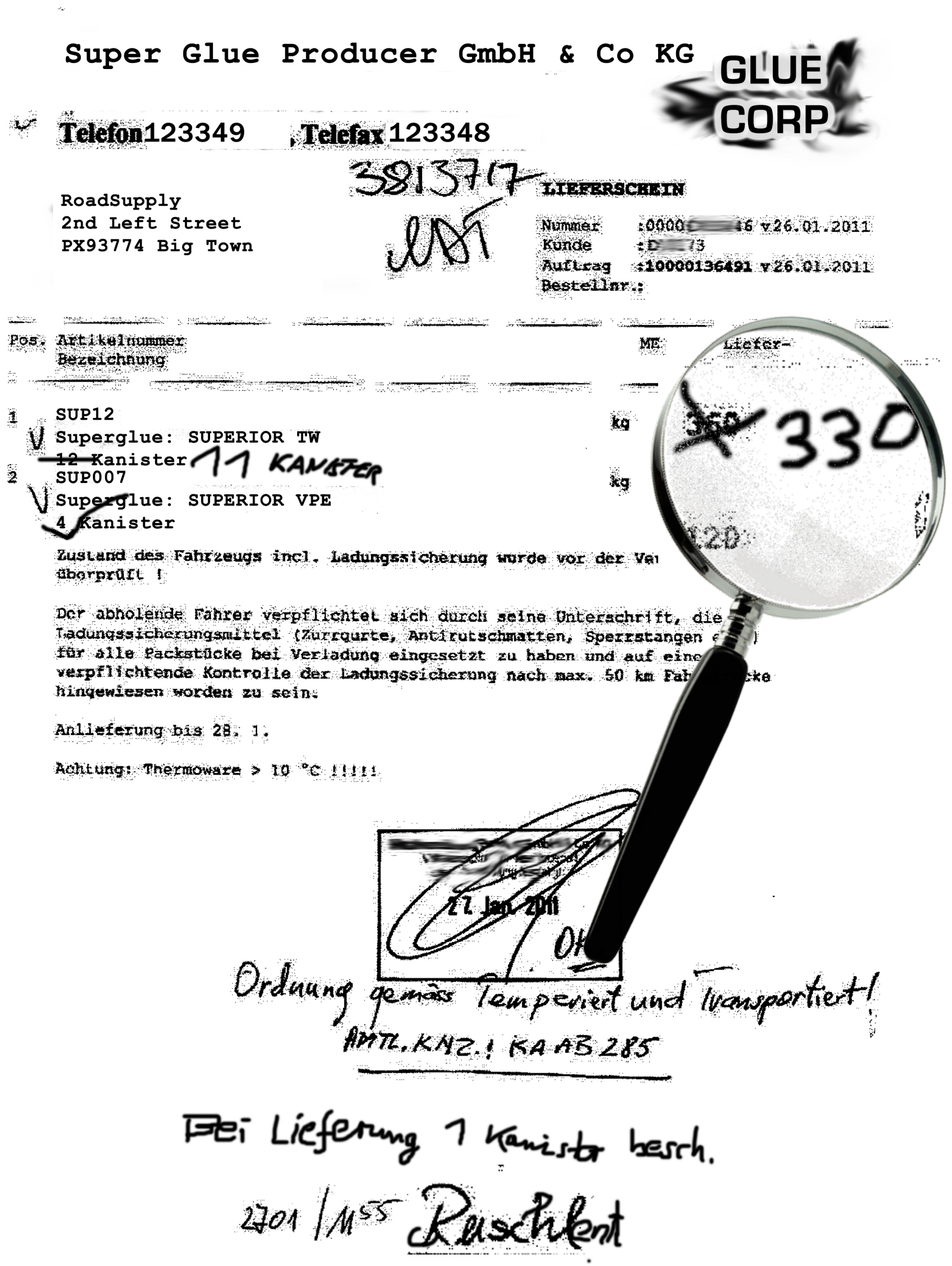
- **Integrity and Authenticity Statements for Partial Data**
- **Legal Compliance, Manage & Verify the Statements**

This research is funded by BMBF (FKZ:13N10966) and ANR (France)

Federal Ministry
of Education
and Research

Why Sanitizable Signatures

- **Notion appeared in literature around 2005 in work of G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik.**
- **Concept also described in earlier works, i.e. Content Extraction Signatures by Steinfeld et al. in 2001**
- **Ever since: Many Schemes**
- **Many Use Cases: i.e. Freight Document**
- **No Implementations**
- **Not applied on a large scale**



Five Sanitizable Signature Schemes

Ateniese et al.'s Scheme:

- Sanitizable Signature based on Chameleon hashes (CH)
- Sanitizer can compute hash collisions, if trapdoor information is known
- Different CH usable within Ateniese scheme :

1. **Krawczyk:** 1st chameleon hash, based on DLP assumption
2. **Ateniese:** ID-based approach
3. **Zhang:** ID-based approach without an UForge-algorithm
4. **Chen:** ID-based approach without the key-exposure-problem

5. Miyazaki et al.'s scheme:

- redactable signature scheme based on commitments
- allows just deletion
- controllable redaction of consecutive sanitizers

XML Signature Syntax and Processing W3C Standard

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<PurchaseOrder>
  <Item id="8492341">
        <Description id="8492340">Video Game</Description>
        <Price>10.29</Price>
  </Item>
  <Signature xmlns=“http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm=“http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <SignatureMethod Algorithm=“http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI=“#xpointer(id('8492340'))">
          <Transforms>
             <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>ABYPTWCzr8F7dXlUKyg1C+tycm4=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>D9hok43bgiRJ9uzp/7A9MA2YZBFuivvzoZTbC(....)DsFCXjtkRxQ==</SignatureValue>
  </Signature>
</PurchaseOrder>
```

Sanitizable XML Signature (W3C standard compliant)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<PurchaseOrder>
  <Item id="8492341">
        <Description id="8492340">Video Game</Description>
        <Price>10.29</Price>
  </Item>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm=„http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="#xpointer(id('8492340'))">
          <Transforms>
             <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.example.org/xmldsig-more#chamhashdisc">
          <ChamHashDiscKeyValue>
              <p>Aa5Mue7ppx2YD7R8KXUqQIKSTSay6jHhWm9L0dxHpL2P</p>
              <q>1yZc93TTjswH2j4UupUgQUkmk1l1GPCtN6Xo7iPSXsc=</q>
              <r>FQrJPkWb0JwiffjrAdbWAoyropQmNohMgEy6ABsvptQ=</r>
              <g>JtqJ1H0NL0Is+6Y797XKQ1hbHc+HYgoGQAkvK8h+q8Y=</g>
              <y>AVwdxMlXF6HIHRHl0r7Xoojb0VoB7ZBP4Dxc83BDDgxG</y>
          </ChamHashDiscKeyValue>
          </DigestMethod>
          <DigestValue>8Xt2AtyvB3Umwf8LlyrGSVnvLc4=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>D9hok43bgiRJ9uzp/7A9MA2YZBFuivvzoZTbC(....)DsFCXjtkRxQ==</SignatureValue>
  </Signature>
</PurchaseOrder>
```

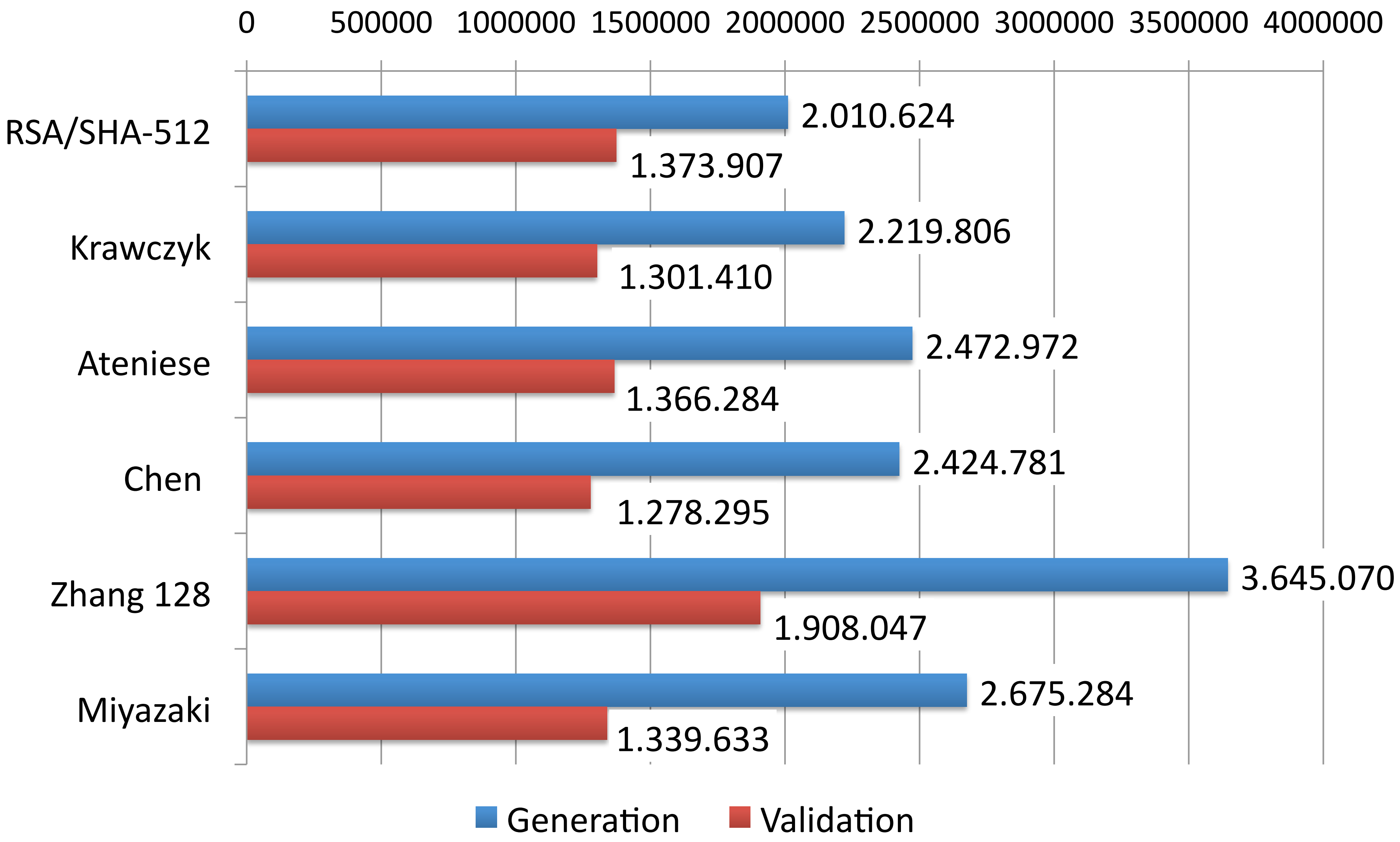
1.
2.
3.
new
4.

Sanitizable XML Signature (W3C standard compliant)

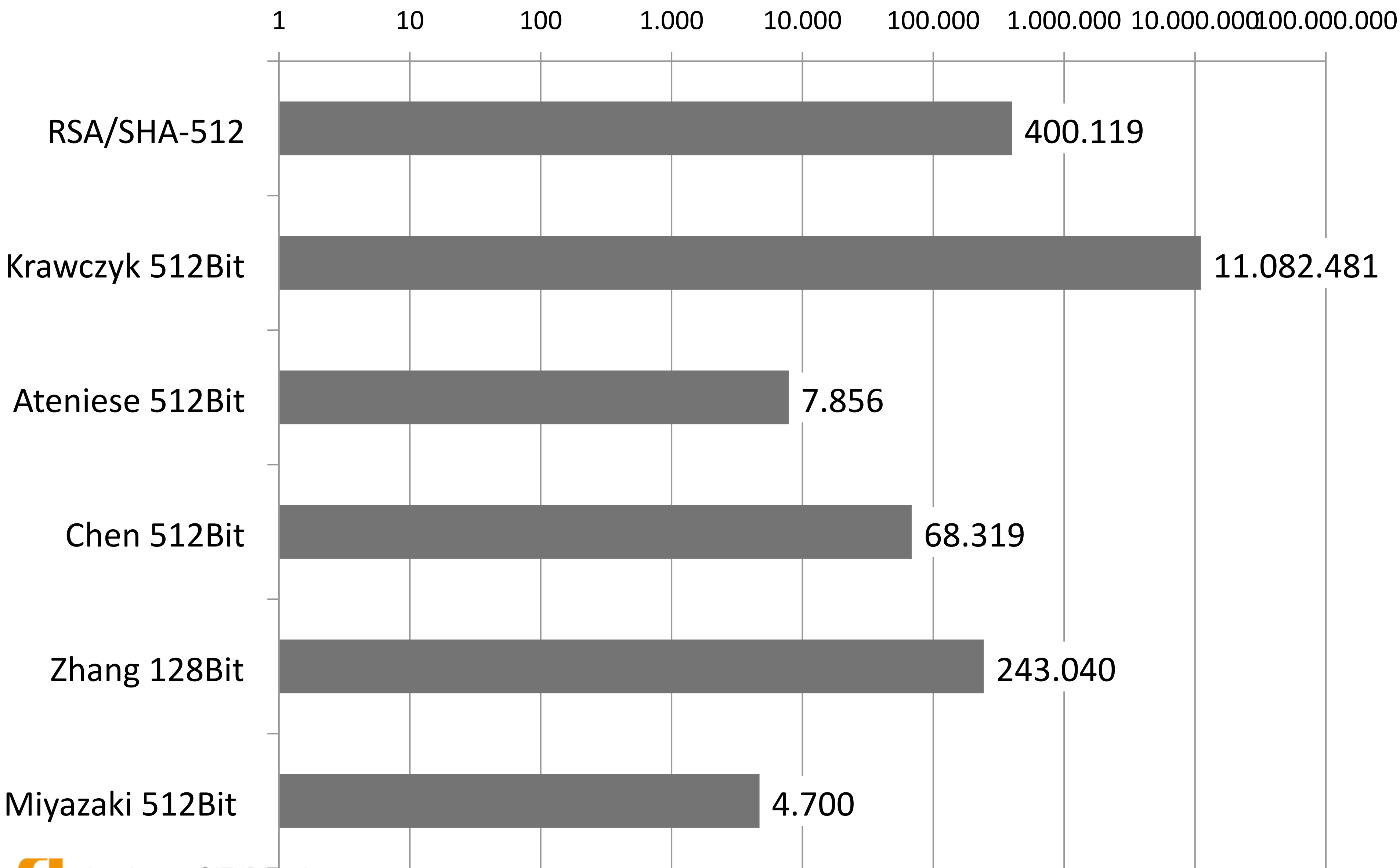
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<PurchaseOrder>
  <Item id="8492341">
       <Description id="8492340">Video Game</Description>
       <Price>10.29</Price>
  </Item>
  <Signature xmlns=“http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
       <CanonicalizationMethod Algorithm=„http://www.w3.org/2001/10/xml-exc-c14n#"/>
       <SignatureMethod Algorithm=“http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
       <Reference URI=“#xpointer(id('8492340'))">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
         </Transforms>
         <DigestMethod Algorithm=“http://www.example.org/xmldsig-more#chamhashdisc">
         </DigestMethod>
         <DigestValue>8Xt2AtyvB3Umwf8LlyrGSVnvLc4=</DigestValue>
       </Reference>
    </SignedInfo>
    <SignatureValue>D9hok43bgiRJ9uzp/7A9MA2YZBFuivvzoZTbC(....)DsFCXjtkRxQ==</SignatureValue>
    <ChamHashDiscKeyValue Id=“#xpointer(id('8492340'))">
            <p>Aa5Mue7ppx2YD7R8KXUqQIKSTSay6jHhWm9L0dxHpL2P</p>
            <q>1yZc93TTjswH2j4UupUgQUkmk1l1GPCtN6Xo7iPSXsc=</q>
            <r>FQrJPkWb0JwiffjrAdbWAoyropQmNohMgEy6ABsvptQ=</r>
            <g>JtqJ1H0NL0Is+6Y797XKQ1hbHc+HYgoGQAkvK8h+q8Y=</g>
            <y>AVwdxMlXF6HIHRHl0r7Xoojb0VoB7ZBP4Dxc83BDDgxG</y>
    </ChamHashDiscKeyValue>
  </Signature>
</PurchaseOrder>
```

1.
2.
3.
4.
new

Performance: Sign & Verify (No Setup) in micro secs.



Performance: Setup incl. KeyPair Generation in micro secs.



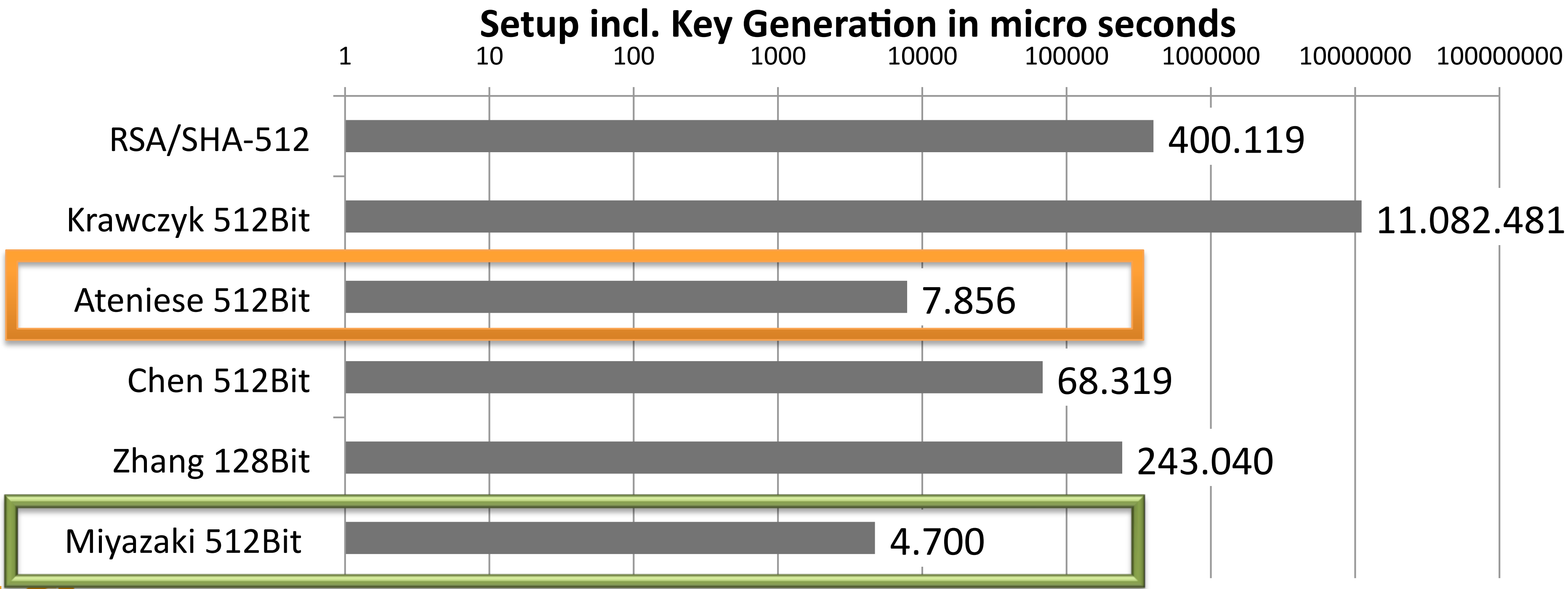
Performance: Summary

excluding key generation & setup:

- **similar runtime as SHA/RSA for signature generation and validation**
- **exception: Zhang et al.'s scheme based on Elliptic Curve Crypto**

including key generation & setup:

- **Overall: Chameleon Hash by Ateniese et al. performs best**



Performance: Summary

excluding key generation & setup:

- similar runtime as SHA/RSA for signature generation and validation
- exception: Zhang et al.'s scheme based on Elliptic Curve Crypto

including key generation & setup:

- Overall: Chameleon Hash by Ateniese et al. performs best

comparison of just one execution is not always enough:

- key exposure problem
- "a forged message and a original message leaks the secret and allows to U-Forge other messages under the same key"

Performance: Summary

excluding key generation & setup:

- **similar runtime as SHA/RSA for signature generation and validation**
- **exception: Zhang et al.'s scheme based on Elliptic Curve Crypto**

including key generation & setup:

- **Overall: Chameleon Hash by Ateniese et al. performs best**

comparison of just one execution is not always enough:

- **suffer from key exposure**
  - **Krawczyk et al.** (needs new key for each message)
- **key exposure problem reduction possible**
  - **Ateniese et al.** (using one time TransactionIDs)
- **key exposure free**
  - **Chen et al. & Zhang et al.**

ID based schemes (Ateniese et al.; Chen et al. & Zhang et al.)

Example: Ateniese Scheme for Chameleon Hashes

Message is split into parts (m_j) , each hashed independently

Message: $m = m_1 || \dots || m_i$

$\mathcal{CH}$: a Chameleon Hash function

k_i : secret key given to the Sanitizer needed for I-Forge
u_i : secret key that is safely discarded

Signature:

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

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

Transparency, Weak Transparency, Strong Transparency

Ateniese et al. define "Transparency":

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

Ateniese et al. divided this into

"Weak Transparency":

(...) the verifier knows exactly which parts of the message
are potentially sanitizable (...)

"Strong Transparency":

(...) the verifier does not know which parts of the message (...)
could potentially be sanitizable.

Properties of Ateniese Scheme for Chameleon Hashes

Message is split into parts (m_j) , each hashed independently

Message: $m = m_1 \,||\, \ldots \,||\, m_i$

$\mathcal{CH}$: a Chameleon Hash function

k_i : secret key given to the Sanitizer needed for I-Forge
u_i : secret key that is safely discarded

Signature:

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

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

Properties: TRANSPARENCY & STRONG TRANSPARENCY

Mixing

Message is split into parts (m_j) , each hashed independently

Message: $m = m_1 \,||\, \ldots \,||\, m_i \;+\; \mathcal{H}(m)$

$\mathcal{CH}$: a Chameleon Hash function

$\mathcal{H}$: a Standard Crypto. Hash

k_i : secret key given to the Sanitizer needed for I-Forge

u_i : secret key that is safely discarded

Signature:

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

$$\sigma = SIGN(d_1||\ldots||d_n||\mathcal{H}(m))$$

Mixing yields: Strong Transparency w/o Transparency

Message is split into parts (m_j) , each hashed independently

Message: $m = m_1 || \ldots || m_i + \mathcal{H}(m)$

$\mathcal{CH}$: a Chameleon Hash function $\mathcal{H}$: a Standard Crypto. Hash

k_i : secret key given to the Sanitizer needed for I-Forge
u_i : secret key that is safely discarded

Signature:

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

$$\sigma = SIGN(d_1||\ldots||d_n||\mathcal{H}(m))$$

Properties: **NO** TRANSPARENCY & STRONG TRANSPARENCY

The Property of Transparency Revisited (1)

Exisiting definitions of Transparency:

- **T always implies WT or ST**
- **WT or ST always implies T**
- **T ⇔ ST or WT**

The Property of Transparency Revisited (1)

Exisiting definitions of Transparency:

- **T always implies WT or ST**
- **WT or ST always implies T**
- **T ⇔ ST or WT**

Practically, a verifier either knows which m_i is potentially sanitizable or he does not:

- $\models$ **(ST or WT)**
- **The result: T is always true**

The Property of Transparency Revisited (1)

Exisiting definitions of Transparency:

- **T always implies WT or ST**
- **WT or ST always implies T**
- **T ⇔ ST or WT**

Practically, a verifier either knows which m_i is potentially sanitizable or he does not:

- **$\models$ (ST or WT)**
- **The result: T is always true**

We found this counter intuitive and the Mixing Example showed no T , but still has ST

The Property of Transparency Revisited (2)

Transparency makes a statement about the detection of a saniti<u>zed</u> document.

Weak and Strong Transparency make statements about the detection of saniti<u>zable</u> <u>sub</u>documents.

Weak and Strong Transparency are independent from Transparency.

Conclusion

1. **JAVA Implementation of 5 Sanitizable Sign. Schemes:**
   - Mostly a tolerable performance penalty over SHA/RSA
   - Full JCA integration as JAVA Crypto Provider
   - Integration into XML Digital Signature Syntax and Processing Standard (W3C)
2. **Mixing Signature / Hash Algorithms is easy and "natural" using XML's `<references>`**
   - Allows fine-grained control over Scheme's properties like Transparency
3. **Property of Transparency is independent from Strong-/Weak-Transparency and has a different scope**

Further Questions? hp@sec.uni-passau.de

1. **JAVA Implementation of 5 Sanitizable Sign. Schemes:**
   - Mostly a tolerable performance penalty over SHA/RSA
   - Full JCA integration as JAVA Crypto Provider
   - Integration into XML Digital Signature Syntax and Processing Standard (W3C)
2. **Mixing Signature / Hash Algorithms is easy and "natural" using XML's `<references>`**
   - Allows fine-grained control over Scheme's properties like Transparency
3. **Property of Transparency is independent from Strong-/Weak-Transparency and has a different scope**


THANK YOU !

The Property of Transparency by Ateniese et al.

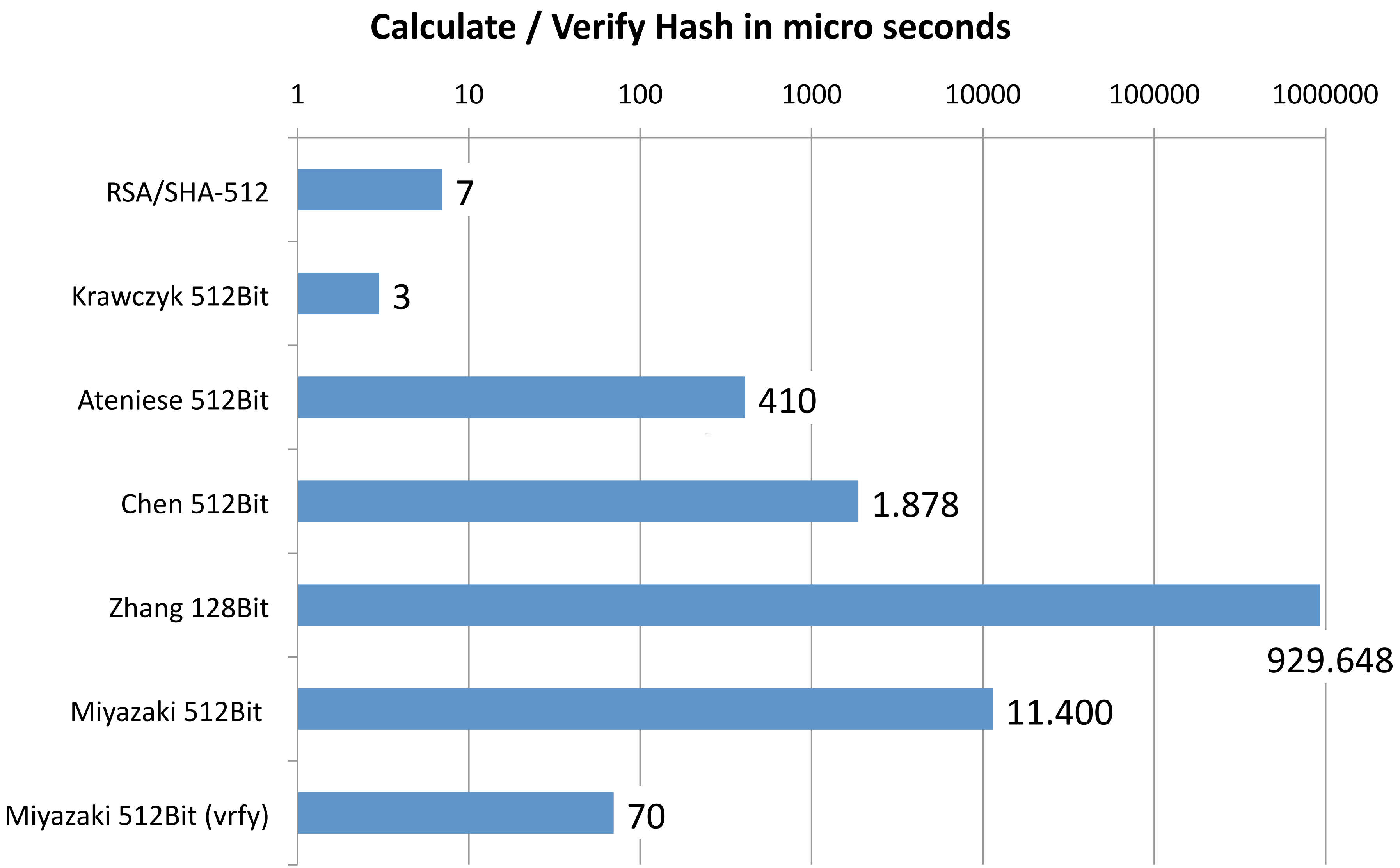
Ateniese et al. define the property of transparency (T) as follows:

> 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]

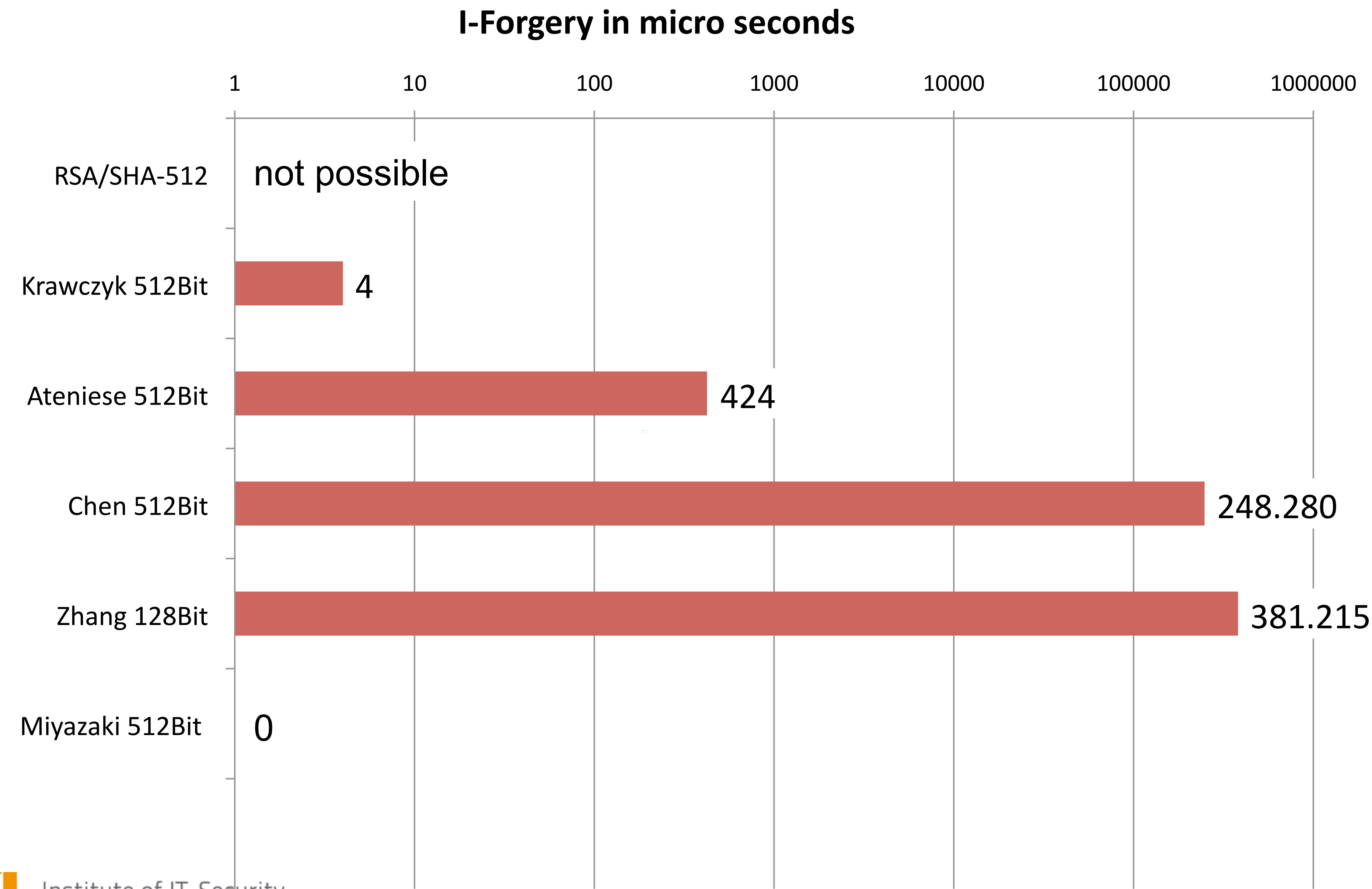
They further divide the property into "weak" (WT) and "strong transparency" (ST):

> 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]

Performance: Calculate / Verify Hash



Performance: Calc. a Forgery using I-Forge Algorithm



Performance: Details

Test Setup:

- **Intel T8300 Dual Core @ 2.40 Ghz and 4 GiB of RAM.**
- **Algorithms coded in JAVA**
- **Not optimized**
- **make heavy use of JAVA's BigInteger class**
- **Integrated into JAVA Cryptographic Framework (JCA) without modifying the JCA**
- **Input: XML File (JAVA JCA Signature Example File)**
  - 1 Reference
  - Fixed size (achievable by applying Standard Crypto. Hash 1st)