Blank Digital Signatures: Optimization and Practical Experiences

David Derler, Christian Hanser, Daniel Slamanig

Blank Digital Signatures: Optimization and Practical Experiences*

David Derler, Christian Hanser, and Daniel Slamanig

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology (TUG), Inffeldgasse 16a, 8010 Graz, Austria
{david.derler|christian.hanser|daniel.slamanig}@tugraz.at

Abstract. Blank Digital Signatures (BDS) [18] enable an originator to delegate the signing rights for a template, containing fixed and exchangeable elements, to a proxy. The proxy is then able to choose one of the predefined values for each exchangeable element and issue a signature for such an instantiation of the template on behalf of the originator. In this paper, we propose optimizations for the BDS scheme from [18] and present a library, integrating this optimized version within the Java Cryptography Architecture and the keying material into X.509 certificates. To illustrate the flexibility of the proposed library, we introduce two proof-of-concept implementations building up on XML and PDF, respectively. Finally, we give a detailed insight in the performance of the protocol and our implementation.

1 Introduction

In contrast to conventional digital signatures, involving a signer and a verifier, proxy-type digital signature schemes are signature schemes involving three parties, namely an originator, a proxy and a verifier. Here, the originator delegates the signing power (for some particular well defined set of messages) to a proxy. The proxy can then sign messages on behalf of the originator. Any verifier, given a message and a corresponding signature, can check whether the proxy has produced the signature on behalf of the originator (authenticity), the integrity of the message and whether the given message is one of the "allowed" messages.

Blank Digital Signatures (BDS) [18] are a special instance of proxy-type digital signatures, allowing an originator to define and issue a signature on a template, containing fixed and exchangeable elements. A designated proxy can then produce signatures for instantiations of this template (messages). More precisely, given a template signature, the proxy creates an instantiation by choosing one of the predefined values for each of the exchangeable elements and issues a signature with respect to the template signature. When verifying this signature, only the message and the corresponding signature is needed, and it is required that

* Part of this work has been supported by the European Commission through project FP7-FutureID, grant agreement number 318424. We thank the anonymous referees for their helpful comments.

the verifier does not learn anything about the unused choices in the exchangeable elements in the template (privacy property).

Blank Digital Signatures give rise to a lot of interesting applications, and, accordingly, the question arises how a BDS scheme would perform in a practical implementation, and to which extent it can be integrated into off-the-shelf cryptographic frameworks such as the Java Cryptography Architecture [26] and key infrastructures such as PKIX [8].

1.1 Our Contribution

In this paper, we propose optimizations for the BDS scheme in [18] and present a full-fledged implementation of this optimized version. Firstly, we briefly revisit the scheme and discuss possible practical applications. Then, we show how the scheme can be modified to use Type-3 pairings instead of the originally proposed Type-1 pairings and introduce optimizations for the encoding of templates. Subsequently, we show how the scheme can be integrated into the Java Cryptography Architecture and how the keying material can be encapsulated within X.509 certificates. Moreover, two possible signature formats, namely an XML and a PDF signature format, are proposed. Finally, timings of our implementation, showing the practical applicability of the BDS scheme, are provided and discussed.

2 Background

We use additive notation for groups, which are always of prime order p. A function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* if for all $c > 0$ there is a k_0 such that $\epsilon(k) < 1/k^c$ for all $k > k_0$. In the remainder of this paper, we use ϵ to denote such a negligible function.

Definition 1 (Bilinear Map:) *A bilinear map (pairing) is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order p. Let P and P' generate $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We require e to be efficiently computable and to satisfy:*

Bilinearity: $e(aP, bP') = e(P, P')^{ab} = e(bP, aP') \quad \forall a, b \in \mathbb{Z}_p$

Non-degeneracy: $e(P, P') \neq 1_{\mathbb{G}_T}$, *i.e.,* $e(P, P')$ *generates* $\mathbb{G}_T$.

If $\mathbb{G}_1 = \mathbb{G}_2$, e is called symmetric *and* asymmetric *otherwise. Asymmetric pairings can be either Type-2 or Type-3 pairings. The difference between Type-2 and Type-3 pairings is that an efficiently computable isomorphism $\Psi : \mathbb{G}_2 \to \mathbb{G}_1$ exists for Type-2 pairings, while for Type-3 pairings such an isomorphism is unknown.*

Definition 2 (t-SDH assumption [5]) *Let p be a prime of bitlength κ, $\mathbb{G}_1$ and $\mathbb{G}_2$ be finite cyclic groups of order p, generated by $P \in \mathbb{G}_1$ and $P' \in \mathbb{G}_2$, respectively, $P = \Psi(P')$, $\alpha \in_R \mathbb{Z}_p^*$ and $t > 0$. Then, for all PPT adversaries $\mathcal{A}$ it holds that*

$$\mathsf{Pr}\left[\left(c, \frac{1}{\alpha + c}P\right) \leftarrow \mathcal{A}(P, (\alpha^i P')_{i=0}^t)\right] \leq \epsilon(\kappa), \text{ where } c \in \mathbb{Z}_p \setminus \{-\alpha\}.$$

In this paper, we concentrate on Type-3 pairings on Barreto-Naehrig curves [4] with embedding degree 12. Thus, elements in $\mathbb{G}_T$ have a bitlength of $12 \cdot \text{bitlength}(p)$. For our setting, we chose a bitlength of 256 bits, leading to a bitlength of 3072bit in $\mathbb{G}_T$. This choice is ideal w.r.t. the comparable strengths proposed by NIST [3], since the discrete logarithm problem should be equally hard in the additive groups $\mathbb{G}_1$, $\mathbb{G}_2$ and in the multiplicative group $\mathbb{G}_T$. In the Type-3 setting, we can use the natural counterpart of the t-SDH assumption, i.e., the co-t-SDH assumption [7, 19].

Definition 3 (co-t-SDH assumption [7, 19]) *Let p be a prime of bitlength κ, $\mathbb{G}_1$ and $\mathbb{G}_2$ be finite cyclic groups of order p, generated by $P_1 \in \mathbb{G}_1$ and $P_2 \in \mathbb{G}_2$, respectively, $\alpha \in_R \mathbb{Z}_p^*$, $i \in \{1, 2\}$ and $t > 0$. Then, for all PPT adversaries $\mathcal{A}$ it holds that*

$$\mathsf{Pr}\left[\left(c, \frac{1}{\alpha + c} P_i\right) \leftarrow \mathcal{A}((\alpha^i P_1)_{i=0}^t, (\alpha^i P_2)_{i=0}^t)\right] \leq \epsilon(\kappa), \text{ where } c \in \mathbb{Z}_p \setminus \{-\alpha\}.$$

2.1 Digital Signature Schemes

A *digital signature scheme* DSS is a triple (DKeyGen, DSign, DVerify) of PPT algorithms. Thereby, DKeyGen is a key generation algorithm that takes a security parameter $\kappa \in \mathbb{N}$ as input and outputs a secret (signing) key sk and a public (verification) key pk. Further, DSign is a (probabilistic) algorithm, which takes a message $M \in \{0,1\}^*$ and a secret key sk as input, and outputs a signature σ. Finally, DVerify is a deterministic algorithm, which takes a signature σ, a message $M \in \{0,1\}^*$ and a public key pk as input, and outputs a single bit $b \in \{\texttt{true}, \texttt{false}\}$ indicating whether σ is a valid signature for M under pk.

A digital signature scheme is required to be *correct*, i.e., for all security parameters κ, all (sk, pk) generated by DKeyGen and all $M \in \{0,1\}^*$ one requires $\mathsf{DVerify}(\mathsf{DSign}(M, \mathsf{sk}), M, \mathsf{pk}) = \texttt{true}$. Additionally, for security one requires existential unforgeability under adaptively chosen-message attacks (EUF-CMA) [16].

2.2 Java Cryptography Architecture

The Java Cryptography Architecture [26] (JCA) constitutes an API, providing standardized access to cryptographic algorithms. Each library that implements this API needs to implement a so-called cryptographic `Provider`, registering the provided algorithm implementations at the JCA. The desired `Provider` is then set by the user of the library, and instances of the algorithm implementations can be obtained using the JCA-provided factories. The primitives we use in this paper are implementations of the `Signature` interface, the `KeyPairGenerator` interface and the `KeyFactory` interface, respectively. The `Signature` interface resembles the DSign and DVerify functionality of a digital signature scheme as discussed above, whereas the `KeyPairGenerator` interface and the `KeyFactory` interface provide methods for conveniently generating and handling keys in general. Using the JCA, entire implementations can be easily exchanged by simply setting another `Provider`.

3 The Blank Digital Signature Scheme

In this section, we introduce the notion of BDS schemes in general, and then give a brief overview of the BDS scheme from [18] (further referred to as BDSS). We discuss the basic building blocks, as well as the principles underlying the signature generation and verification. Since this paper lays focus on the practical aspects of the BDSS, we keep this section quite informal and refer the reader to [18] for more formal definitions.

A BDS scheme allows an *originator* to designate the signing rights for a certain template to a *proxy*. A template $\mathcal{T}$, thereby, is a sequence of non-empty sets of bitstrings T_i. Depending on the cardinality of the respective set, such sets are either called fixed or exchangeable elements, i.e., fixed elements contain exactly one bitstring, whereas exchangeable elements contain $k > 1$ distinct bitstrings. More formally, we have:

$$\mathcal{T} = (T_1, T_2, \ldots, T_n),\ T_i = \{M_{i_1}, M_{i_2}, \ldots, M_{i_k}\}.$$

The template length is defined as the sequence length n of the template, while the template size $|\mathcal{T}|$ is defined as $|\mathcal{T}| = \sum_{i=1}^{n} |T_i|$. Furthermore, each template is assigned a unique identifier $id_{\mathcal{T}}$. Once the template is defined, the *originator* issues a signature on $\mathcal{T}$ for a particular *proxy*. Based on this so called template signature, the designated *proxy* can choose concrete values for each exchangeable element (fixed elements stay fixed) and compute a so called instance signature on this message $\mathcal{M} = (M_i)_{i=1}^{n}$. With the instance signature at hand, anyone is able to verify the validity of the instance signature and the designation.

Besides the usual *correctness* property, a BDS scheme provides *unforgeability*, *immutability* and *privacy*. Informally, these properties are defined as follows. *Unforgeability* requires that, without knowledge of the secret keys, it is intractable to (existentially) forge template or instance signatures. *Immutability* essentially models a stronger adversary in the unforgeability setting, i.e., additionally covers adversaries knowing the signing key of the proxy. Finally, *privacy* requires that it is intractable (for outsiders) to determine template elements (except the ones revealed by instantiations).

3.1 Applications

Basically, a BDS scheme enables an originator to hand over a signed form (template), containing fixed and exchangeable elements, to a proxy being designated to sign an arbitrary instance of this form, i.e., a filled in form, on behalf of the originator. Figure 1 illustrates a sample template running through a BDS protocol execution. As shown in this figure, it is also possible to encode yes-/no-choices within a template by simply encoding yes and no in an exchangeable element.

In particular, a BDS scheme is applicable to any contract, which requires to leave a few choices open to an intermediary party, while the rest of the content is fixed. For instance, it would be thinkable that a broker makes a business deal on behalf of a client, using a template, previously defined and signed by

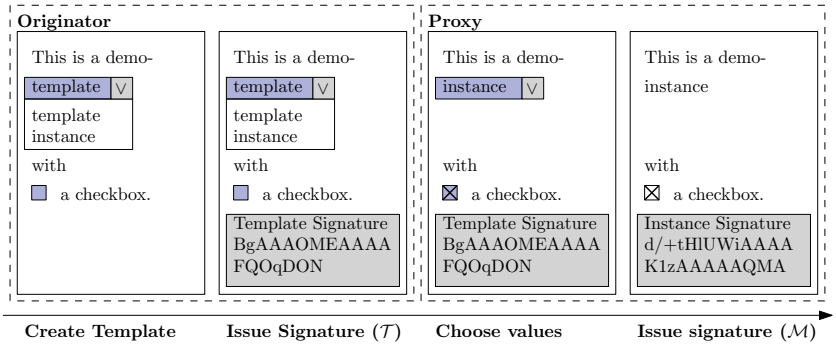


Fig. 1: Schematic View of the BDS Scheme

the client [18]. Thereby, it can be of importance that the unused choices of the template do not get revealed upon verification of an instance signature, which is ensured by the *privacy property* of BDS schemes. Other applications cover various fields, among others, any types of digital reports like lab reports in healthcare, or authorized (public) tender forms, questionnaires and application forms in the eGovernment field.

3.2 The BDSS

From a technical point of view, the BDSS builds up on standard digital signature schemes and a modified version of the polynomial commitments proposed by Kate et al. [24].[1] Using these polynomial commitments necessitates a unique encoding, mapping templates and messages to polynomials in the polynomial ring $\mathbb{Z}_p[X]$. For the rest of this paper, let $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$ be a secure cryptographic hash function. The BDSS uses the following (unique) encoding for a template $\mathcal{T}$, which is denoted by $t(X) \in \mathbb{Z}_p[X]$:

$$t(X) \leftarrow \prod_{i=1}^{n} \prod_{M \in T_i} (X - H(M||id_{\mathcal{T}}||i)).$$

An encoding $m(X) \in \mathbb{Z}_p[X]$ of a message $\mathcal{M}$ looks as follows:

$$m(X) \leftarrow \prod_{i=1}^{n} (X - H(M_i||id_{\mathcal{T}}||i)).$$

Finally, the so-called complementary message polynomial $\overline{m}(X)$ is defined such that $t(X) = m(X) \cdot \overline{m}(X)$ holds. More precisely, $\overline{m}(X)$ contains all factors which are contained in $t(X)$ but not in $m(X)$. For the rest of this paper, we use $\mathcal{C}_{\mathcal{T}}$, $\mathcal{C}_{\mathcal{M}}$ and $\mathcal{C}_{\overline{\mathcal{M}}}$ to denote the (polynomial) commitments to the encodings of templates, messages and complementary-messages, respectively. The commitments used in

[1] Note that this polynomial commitment variant has later been formalized in [19].

the BDSS are unconditionally hiding and computationally binding and due to the nature of the commitments (they are instantiated within bilinear groups), it holds that

$$e(\mathcal{C}_{\mathcal{T}}, P) = e(\mathcal{C}_{\mathcal{M}}, \mathcal{C}_{\overline{\mathcal{M}}}).$$

The BDSS defines five algorithms, which we briefly introduce subsequently. We assume the public parameters pp generated in KeyGen to be an implicit input to all other algorithms. Furthermore, we assume that both, the originator and the proxy are already in possession of a keypair for a conventional DSS.

KeyGen: This algorithm takes a security parameter κ and an upper bound t for the template size. It chooses two groups $\mathbb{G}_1, \mathbb{G}_T$ of the same prime order p (with $\log_2 p = \kappa$), generated by P, having a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, a secure cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ and a random $\alpha \in \mathbb{Z}_p^*$. Finally, it outputs the public parameters $\mathsf{pp} = (H, \mathbb{G}_1, e, p, (\alpha^i P)_{i=0}^t)$.[2]

Sign: This algorithm takes a template $\mathcal{T}$ of length n, the signing key of the originator $\mathsf{sk_O}$ and the verification key of the proxy $\mathsf{pk_P}$. It computes the commitment $\mathcal{C}_{\mathcal{T}}$ to $\mathcal{T}$, $\mathcal{C} = e(\mathcal{C}_{\mathcal{T}}, P)$ and $\tau = \mathsf{DSign}(id_{\mathcal{T}}||\mathcal{C}||n||\mathsf{pk_P}, \mathsf{sk_O})$ and outputs $\sigma_{\mathcal{T}} = (id_{\mathcal{T}}, \mathcal{C}, n, \tau)$ together with a private instantiation key for the proxy $\mathsf{sk_P^{\mathcal{T}}}$ (required for recomputing the commitment).

Verify$_{\mathcal{T}}$: This algorithm takes a template $\mathcal{T}$, a template signature $\sigma_{\mathcal{T}}$, the verification keys of the originator ($\mathsf{pk_O}$) and the proxy ($\mathsf{pk_P}$), as well as $\mathsf{sk_P^{\mathcal{T}}}$. It computes $\mathcal{C}_{\mathcal{T}}$ and $\mathcal{C} = e(\mathcal{C}_{\mathcal{T}}, P)$, and outputs the result of $\mathsf{DVerify}(\sigma_{\mathcal{T}}, id_{\mathcal{T}}||\mathcal{C}||n||\mathsf{pk_P}, \mathsf{pk_O})$.

Inst: This algorithm takes a template $\mathcal{T}$ with corresponding message $\mathcal{M}$, a template signature $\sigma_{\mathcal{T}}$, the signing key of the proxy $\mathsf{sk_P}$ and the instantiation key $\mathsf{sk_P^{\mathcal{T}}}$. It computes $\mathcal{C}_{\overline{\mathcal{M}}}$ and $\mu = \mathsf{DSign}(\tau||\mathcal{C}_{\overline{\mathcal{M}}}||\mathcal{M}, \mathsf{sk_P})$ and returns $\sigma_{\mathcal{M}} = (\mu, \mathcal{C}_{\overline{\mathcal{M}}}, \mathcal{M}, \sigma_{\mathcal{T}})$.

Verify$_{\mathcal{M}}$: This algorithm takes $\mathcal{M}$, $\sigma_{\mathcal{M}}$ and the verification keys of the originator ($\mathsf{pk_O}$) and the proxy ($\mathsf{pk_P}$) and computes $\mathcal{C}_{\mathcal{M}}$ from $\mathcal{M}$. Then, it checks whether $\mathsf{DVerify}(\tau, id_{\mathcal{T}}||\mathcal{C}||n||\mathsf{pk_P}, \mathsf{pk_O}) = \texttt{true}$ and $\mathsf{DVerify}(\mu, \tau||\mathcal{C}_{\overline{\mathcal{M}}}||\mathcal{M}, \mathsf{pk_P}) = \texttt{true}$ holds. If so, it checks whether the number of elements in the message is equal to n, whether there is exactly one element at each position in the message and whether $e(\mathcal{C}_{\mathcal{M}}, \mathcal{C}_{\overline{\mathcal{M}}}) = \mathcal{C}$. On success it returns `true` and `false` otherwise.

4 Tweaks and Optimizations

Since the BDSS is designed for Type-1 pairings, we need to modify the scheme to make it compatible with much more efficient Type-3 pairings. In this section we discuss these modifications, together with an optimization regarding the encoding of templates and messages to reduce the degree of the encoding polynomials.

[2] Note that these parameters are required for computing the polynomial commitments.

4.1 Using Type-3 Pairings

The authors of [18] informally suggested that the scheme can be used with Type-3 pairings by duplicating some of the points in the system-wide parameters, i.e., some points in $\mathbb{G}_1$ also have to be mapped to points in $\mathbb{G}_2$. In the following, we discuss the necessary modifications in detail. For all these modifications it is crucial that the counterpart Q' in $\mathbb{G}_2$, of a point Q in $\mathbb{G}_1$, contains the same discrete logarithm as the point Q, i.e., $Q = aP$ and $Q' = aP'$ for $a \in \mathbb{Z}_p$.

Currently, the system-wide parameters pp of the BDS scheme contain a sequence $\mathcal{P} = (\alpha^i P)_{i=0}^t$ of multiples of a point P, with t being the maximal template size. For Type-3 pairings, the sequence has to be extended with the same multiples of a point $P' \in \mathbb{G}_2$, i.e., $\mathcal{P}' = ((\alpha^i P)_{i=0}^t, (\alpha^i P')_{i=0}^t)$.

In the subsequent protocol steps, one has to choose the appropriate representative of the required point, i.e., the representative in $\mathbb{G}_1$ or $\mathbb{G}_2$. Additionally, in the verification step of the message ($\mathsf{Verify}_{\mathcal{M}}$) the pairing $e(\mathcal{C}_{\mathcal{M}}, \mathcal{C}_{\overline{\mathcal{M}}})$ is evaluated. Thus, in the instantiation step (Inst), the commitment to the complementary message polynomial $\mathcal{C}_{\overline{\mathcal{M}}}$ needs to be computed in $\mathbb{G}_2$.[3] Using this modification, the computation of $\mathcal{C}_{\overline{\mathcal{M}}}$ is the only remaining computation which requires operations in $\mathbb{G}_2$. Thus, it seems to be impossible to find further optimizations based on moving computations from one group to the other.

It is easy to see that switching to the Type-3 setting does not influence the security of the scheme. The original BDSS [18] was proven secure under the t-SDH assumption. Using the co-t-SDH assumption, the security proof of the modified BDSS is (up to the extended problem instance) equivalent to the original proof in [18], and, thus, using Type-3 pairings does not influence the security of the scheme.

4.2 Aggregating Fixed Elements

An important optimization can be based on the reduction of the degree of the encoding polynomials by aggregation. The idea behind the aggregation of the fixed elements is the observation that in the originally proposed BDSS encoding, each fixed element corresponds to one factor in the encoding polynomials. The scheme does, however, not require this separate encoding. Thus, we can simply aggregate the fixed elements within one factor of the encoding polynomials by concatenating the identifier of the template, the messages and the positions of the messages in the template as follows:

$$m_i = M_i||i, \;\; M = m_1||m_2||...||m_u$$
$$m_{\text{fixed}}(X) = X - H(id_{\mathcal{T}}||M).$$

This reduces the degree of the encoding polynomials, and, thus, also the computation times. Note that this optimization also enables the reduction of the size

[3] We note that it would also be possible to compute $\mathcal{C}_{\mathcal{M}}$ in $\mathbb{G}_2$ and evaluate the pairing $e(\mathcal{C}_{\overline{\mathcal{M}}}, \mathcal{C}_{\mathcal{M}})$ upon $\mathsf{Verify}_{\mathcal{M}}$. Then, $\mathcal{C}_{\overline{\mathcal{M}}}$ would still be computed in $\mathbb{G}_1$. However, our goal is to make $\mathsf{Verify}_{\mathcal{M}}$ as fast as possible, and, thus opt for the former option (observe that computations in $\mathbb{G}_2$ are more expensive than computations in $\mathbb{G}_1$).

of the system parameters pp, i.e., pp is no longer dependent on the number of fixed elements. Subsequently, we analyze the security of these modification.

Proof. In the original construction [18], every fixed element represents a factor in the template encoding polynomial and in further consequence in every message encoding polynomial. The modification proposed here integrates all fixed elements into a single factor, which reduces the degree of the respective polynomials. Now, we have to show that this has no impact on the security of the construction. Our argumentation is as follows. Using one factor for the fixed elements in the modified version can be seen as the original construction using only a single fixed element in the template. Therefore, the construction as such still remains secure. What remains to show, however, is that the modified encoding does not influence the correctness (signature soundness) and the unforgeability as well as immutability, respectively.

In this context, signature soundness essentially says that, given a template signature $\sigma_{\mathcal{T}}$ for some template $\mathcal{T}$, the probability that this signature will verify for any $\mathcal{T}' \neq \mathcal{T}$ is negligible in the security parameter κ. To achieve this (for fixed elements), one would need to find

$$H(id_{\mathcal{T}} \| m_{i_1} \| i_1 \| \ldots \| m_{i_u} \| i_u) = H(id_{\mathcal{T}'} \| m'_{i'_1} \| i'_1 \| \ldots \| m'_{i'_{u'}} \| i'_{u'}),$$

which is clearly intractable if H is collision resistant. The same argumentation holds for unforgeability and immutability (cases **T1, M1** [18]), where the problem is to find a second preimage for $H(id_{\mathcal{T}} \| m_{i_1} \| i_1 \| \ldots \| m_{i_u} \| i_u)$. □

5 Implementation in JAVA and Integration into the JCA

In this section, we provide an in-depth description of the implementation related aspects of the optimized BDSS. Our design is based on the observation that the signing and verification algorithms for both, templates and messages, can be interpreted as conventional signature algorithms with special types of messages. This means that one can use a standard signature API, such as the one provided by the Java Cryptography Architecture (JCA) [26], to obtain an easy to use implementation. Furthermore, X.509 provides, among others, a convenient and well-established method to ensure key authenticity and integrity. Besides, also methods for revocation checking are provided [8]. Thus, we follow this approach and integrate the keying material within X.509 certificates.

Finally, we propose two container formats, i.e., XML and PDF, encapsulating the templates and messages, respectively, together with their corresponding signatures.

5.1 Overview

Figure 2 links the BDSS algorithms to the parties performing the respective computations and provides an overview of the required interaction during a usual workflow (note that all algorithms are non-interactive). The gray boxes in

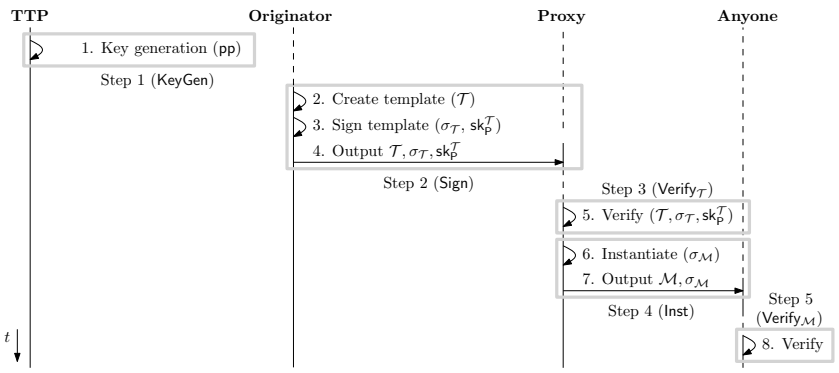


Fig. 2: BDS Scheme Computation Steps

the figure logically group consecutive computation steps to units with defined input and output. For the sake of simplicity, we omitted the visualization of the distribution of the system parameters pp by the trusted third party (TTP). We, however, assume that the TTP provides means for retrieving pp in an authentic manner. For instance, our default implementation encapsulates pp within an X.509 certificate (cf. Section 5.2). To provide maximum flexibility, the library relies on a generic interface for accessing the TTP, and, consequently, our library is not bound to one fixed TTP implementation. From a JCA point of view, the generation of pp is wrapped in a `KeyPairGenerator` implementation and is, thus, conveniently usable by arbitrary TTP implementations.

The subsequent steps, i.e., Step 2-5 in Figure 2, are packed into two JCA `Signature` implementations, namely the `BDSSTemplateSignature` (Step 2 and 3) and the `BDSSInstanceSignature` (Step 4 and 5). To be compatible with the JCA `Signature` interface, we override the `engineSetParameter` method. This way, it is possible to supply so called `AlgorithmParameterSpec` implementations containing the additionally required parameters for executing the protocol. Furthermore, the API assumes that the signing and verification algorithms operate on arrays of bytes. Thus, we (de-)serialize the respective inputs to preserve their structure (cf. Section 5.2). Listing 1 provides an example for obtaining a BDSS signature on a template. The `BDSSInstanceSignature` can be used in a similar way and is therefore omitted.

```
1  Signature signature = Signature.getInstance("BDSSTemplateSignature");
2  TemplateSignParamSpec p = new TemplateSignParamSpec(pp, pkP);
3  signature.setParameter(p);
4  signature.initSign(skO); // set sign mode
5  signature.update(template.serialize()); // add data
6  byte[] templateSignature = signature.sign(); // sign
7
8  TemplateVerifyParamSpec pv = new TemplateVerifyParamSpec(pp, pkP);
9  signature.setParameter(pv);
10 signature.initVerify(pkO); // set verify mode
11 signature.update(template.serialize()); // add data
12 boolean success = signature.verify(templateSignature); // verify
```

Listing 1: Java Code to Obtain a BDS Template Signature

Note that the returned template signature also contains the instantiation key $\mathsf{sk}_{\mathsf{P}}^{\mathcal{T}}$, which needs to be removed when a use case requires to publish the template.

Also note that if the privacy property of the BDSS is required, a secure transmission of the output of Step 2 in Figure 2 is inevitable. Thus, our library provides means for ECIES (see e.g., [22]) encryption and decryption.

5.2 Encoding and Key Representation in X.509

As mentioned before, it is required to (de-)serialize the templates and messages with corresponding signatures to be compatible with the API of the JCA. Consequently, a compact encoding with minimal overhead is desired to keep the transmission times low. We use a unique encoding, similar to the BER/DER [21] encoding of ASN.1 [20] and provide means for serialization and deserialization.

It also turns out that this encoding is useful to integrate the keying material as public key info into X.509 certificates [8]. To (re-)extract the serialized keys from the public key info, our Java cryptographic provider provides the appropriate `KeyFactory` implementations (performing the deserialization).

To bring the (signed) templates and messages into a user friendly form, e.g., to support users to conveniently fill in a templates, we introduce two container formats in the remainder of this section. For both formats, we follow the approach that the templates and messages are included in a human readable form, whereas the signatures are serialized using our encoding from above.

5.3 Defining an XML Signature Format

To use XML, we added Java annotations for XML binding (JAXB), as defined in [13], to the classes serving as input-/output-containers. These annotations, together with the appropriate XML schema allow to conveniently marshal/unmarshal Java objects to/from XML using the routines provided by the Java platform. Listing 2 and Listing 3 show the proposed signature format, with "?" and "+" denoting the multiplicity of the tags, i.e., "?" means at most once, whereas "+" means at least once.

```
<template id="...">
  (<templateentry>
    (<message type="exch"|"fix" length="[Integer]">
      <text>[String]</text>
    </message>)+
  </templateentry>)+
  (<signature>
    <signaturevalue>[Base64 encoded string]</signaturevalue>
    (<keyId>[String]</keyId>)?
    (<ttpcert>[Base64 encoded string]</ttpcert>)?
    (<originatorcert>[Base64 encoded string]</originatorcert>)?
    (<proxycert>[Base64 encoded string]</proxycert>)?
  </signature>)?
</template>
```

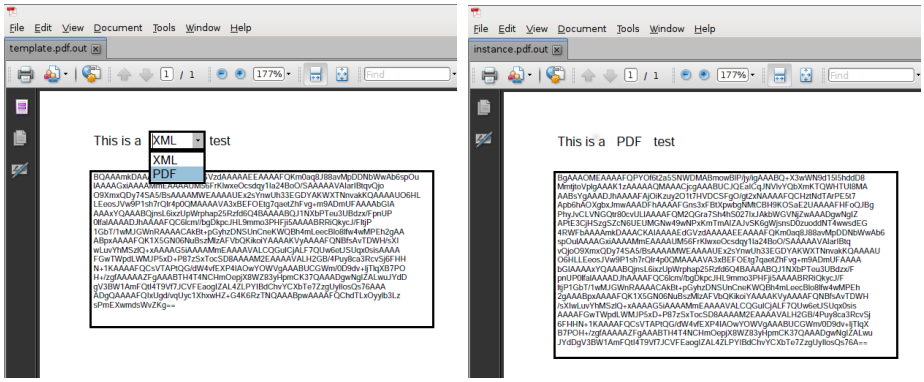
Listing 2: BDS Template Format

```
<instance id="...">
  (<message type="exch"|"fix" length="[Integer]">
    <text>[String]</text>
  </message>)+
  (<signature>
    <signaturevalue>[Base64 encoded string]</signaturevalue>
    (<keyId>[String]</keyId>)?
    (<ttpcert>[Base64 encoded string]</ttpcert>)?
    (<originatorcert>[Base64 encoded string]</originatorcert>)?
    (<proxycert>[Base64 encoded string]</proxycert>)?
  </signature>)?
</instance>
```

Listing 3: BDS Message Format

5.4 Using PDF as Signature Format

Signable PDF forms seem to be an essential application of BDSS. Thus, a proof-of-concept implementation using PDF as container format is introduced subsequently. Thereby, our library provides means to create, sign and verify templates and messages in PDF format. Furthermore, signed templates can directly be filled in in the same way as conventional PDF forms using a standard PDF reader. Figure 3 shows a sample template and a corresponding message, both containing a signature.



(a) Signed Template (b) Signed Message

Fig. 3: BDS PDF Signature Format

6 Performance Evaluation

In this section, we provide an overview of the performance of our proof-of-concept implementation. For the timings, we use the BNPairings library [15] for computing the optimal Ate pairing [30] on BN curves [4] with 256 bit group size and an

embedding degree of 12. As conventional digital signatures we use ECDSA [14,17] with the NIST P-224 curve [14]. The timings were performed using a *single* core on a *Lenovo ThinkPad T420s* with an *Intel Core i5 2540M* with 2.6/3.3 GHz and 8 GB of RAM. On the software side Java 1.7.0_55 was used on top of Ubuntu 14.04/amd64.

We measure the execution time of the four protocol steps (Step 2-5 in Figure 2) for different template sizes and template compositions. To isolate timing-related influences, e.g., the garbage collector, each timing represents the mean of 100 consecutive runs. Table 1 shows the computation times for template sizes ranging from 3 to 1000. To illustrate the influence of the distribution of the element types on the timing, we provide timings for two different element type distributions. Note that it does not make sense to choose a template with less than 50% of exchangeable elements, since templates are always chosen minimal, i.e., there are no two fixed elements next to each other.[4] The used element type distribution is indicated by the percentage values in the top row of Table 1. Figure 4 gives an overview of the computation time with increasing template size. As expected, the computation times depend heavily on the degree of the

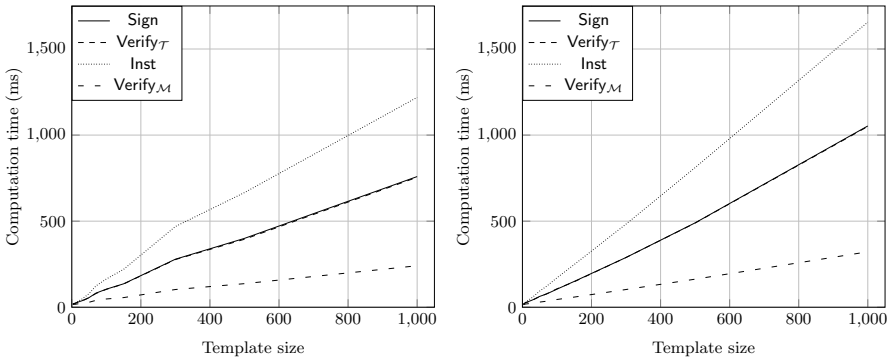
	50% fixed				**33% fixed**			
	Template		**Message**		**Template**		**Message**	
$\|\mathcal{T}\|$	Sign	Verify	Inst	Verify	Sign	Verify	Inst	Verify
3	20	19	18	17	18	18	16	14
5	21	20	16	17	23	22	23	17
10	23	23	24	17	28	27	31	19
15	28	27	31	19	31	30	38	20
30	38	37	48	23	43	42	59	24
50	56	55	79	29	63	62	94	30
70	82	80	124	40	77	78	122	35
100	105	104	164	48	105	107	171	45
150	136	135	220	56	150	148	248	59
300	279	277	469	103	289	289	483	103
500	400	395	666	137	490	489	811	163
1000	759	755	1219	241	1053	1050	1656	322

Table 1: Timings for Various Template Sizes in Milliseconds

encoding polynomials. Consequently, having more fixed elements, for the same size $|\mathcal{T}|$, results in a lower degree polynomial – using our trick from Section 4.2 – and corresponding shorter computation times.

[4] For some applications it could make sense to place two or more exchangeable elements next to each other, which would allow to encode ranges. For instance, all three digit numbers could be modeled by three exchangeable elements, each containing the numbers from 0 to 9.

In practice, most forms will contain less than 100 elements, which leads to computation times of less than 180ms for each step. This is perfectly acceptable for practical use.



(a) 50% fixed (b) 33% fixed

Fig. 4: Computation Times in Relation to the Template-Size and the Element Type Distribution.

Finally, observe that BDS allow to quite straightforwardly define the template in a way, which enables similar functionalities as redactable/sanitizable signatures [2, 6, 23, 25, 29]. Although, such an application of BDS is not considered in this paper (it has been done in [28] as a replacement for redactable signatures as used in [31]), we conclude that – due to its good performance – our BDSS implementation might also be an alternative to implementations of redactable/sanitizable signatures (e.g., [27]) in certain settings.

7 Conclusion and Future Work

In this paper, we proposed an optimization regarding the template and message encoding of the BDSS and modified it to use much more efficient Type-3 pairings. We introduced a JCA-based interoperable framework for the BDSS, providing an easy to use API. To illustrate the capabilities, concerning the integration into other applications, two signature formats were proposed. Moreover, we gave an overview of the performance of the scheme and our implementation. Meanwhile, our implementation based on the XML signature format has been integrated into the *FutureID eSignServices* framework [1] – a flexible framework for signature generation and validation.

The execution times presented in Section 6 are totally practical, since in most scenarios it can be expected that templates will have a template-size of less than 100, leading to computation times of less than 180ms for arbitrary

template constellations. This shows that the BDSS is fully feasible for practical use. For further details on our BDSS implementation and optimization, we refer the reader to [9].

Finally, there are some points we leave open for future work. Quite recently, a black-box construction of BDS from non-interactive anonymous credentials was presented in [10]. It would, thus, be interesting to compare the performance of an implementation of this construction to our implementation. Another interesting step would be to increase the practical usability of our implementation by integrating the BDSS within a plug-in of a PDF reader. Furthermore, we do not expect any problems when integrating the XML signatures proposed in Section 5.3 into other XML signature formats such as XMLDSig [11] or the various types of XML Advanced Electronic Signatures [12]. The latter format would, in turn, enable long term signature validation, which could be of particular interest for BDSS signed contracts.

References

1. FutureID Project, `http://www.futureid.eu`
2. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable Signatures. In: ESORICS. LNCS, vol. 3679, pp. 159–177. Springer (2005)
3. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for Key Management - Part 1: General (Revision 3). In: NIST Special Publication (2012)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Selected Areas in Cryptography. pp. 319–331. LNCS, Springer (2005)
5. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Advances in Cryptology - EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer (2004)
6. Brzuska, C., Busch, H., Dagdelen, Ö., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A., Poettering, B., Schröder, D.: Redactable signatures for tree-structured data: Definitions and constructions. In: ACNS. LNCS, vol. 6123, pp. 87–104. Springer (2010)
7. Chatterjee, S., Menezes, A.: On cryptographic protocols employing asymmetric pairings - The role of ψ revisited. Discrete Applied Mathematics 159(13), 1311–1322 (2011)
8. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (May 2008)
9. Derler, D.: On the Optimization of two Recent Proxy-Type Digital Signature Schemes and their Efficient Implementation in Java. Master's thesis, Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology (2013)
10. Derler, D., Hanser, C., Slamanig, D.: Privacy-Enhancing Proxy Signatures from Non-interactive Anonymous Credentials. In: Data and Applications Security and Privacy XXVIII, LNCS, vol. 8566, pp. 49–65. Springer (2014), `http://dx.doi.org/10.1007/978-3-662-43936-4_4`
11. Eastlake, D., Reagle, J., Solo, D.: XML-Signature Syntax and Processing (2002), W3C Recommendation
12. European Telecomunications Standards Institute: Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES); ETSI TS 101 903 (2010)

13. Fialli, J., Vajjhala, S.: Java Architecture for XML Binding (JAXB) 2.0. Java Specification Request (JSR) 222 (October 2005)
14. Gallagher, P., Foreword, D.D., Director, C.F.: FIPS PUB 186-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS) (2009)
15. Geovandro, C.C.F.P., Barreto, P.S.L.M.: bnpairings - A Java implementation of efficient bilinear pairings and elliptic curve operations. Public Google code project at: `https://code.google.com/p/bnpairings/` (5 November 2012)
16. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM J. Comput. 17(2), 281–308 (1988)
17. Hanser, C.: IAIK ECCelerate SDK 2.51 (2014)
18. Hanser, C., Slamanig, D.: Blank Digital Signatures. In: 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (AsiaCCS). Full Version: Cryptology ePrint Archive, Report 2013/130. pp. 95–106. ACM (2013)
19. Hanser, C., Slamanig, D.: Structure-Preserving Signatures on Equivalence Classes and their Application to Anonymous Credentials. In: Advances in Cryptology - ASIACRYPT 2014. (Full version Cryptology ePrint Archive Report 2014/705). LNCS, vol. 8873, pp. 491–511. Springer (2014)
20. International Telecommunication Union: Information Technology — Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. ITU-T Recommendation X.680 (July 2002)
21. International Telecommunication Union: Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). ITU-T Recommendation X.690 (July 2002)
22. Information Technology - Security Techniques - Encryption Algorithms - Part 2: Asymmetric Ciphers (2006)
23. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic Signature Schemes. In: CT-RSA. LNCS, vol. 2271, pp. 244–262. Springer (2002)
24. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: ASIACRYPT. LNCS, vol. 6477, pp. 177–194. Springer (2010)
25. Miyazaki, K., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H., Tezuka, S., Imai, H.: Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. IEICE Transactions 88-A(1), 239–246 (2005)
26. Oracle: Java™ Cryptography Architecture (JCA) Reference Guide. `http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html`
27. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable signatures in XML Signature - performance, mixing properties, and revisiting the property of Transparency. In: ACNS. LNCS, vol. 6715. Springer (2011)
28. Slamanig, D., Stranacher, K., Zwattendorfer, B.: User-Centric Identity as a Service-Architecture for eIDs with Selective Attribute Disclosure. In: 19th ACM Symposium on Access Control Models and Technologies (SACMAT 2014). pp. 153 – 163. ACM (2014)
29. Steinfeld, R., Bull, L., Zheng, Y.: Content Extraction Signatures. In: ICISC. LNCS, vol. 2288, pp. 285–304. Springer (2001)
30. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory 56(1), 455–461 (2010)

31. Zwattendorfer, B., Slamanig, D.: On Privacy-Preserving Ways to Porting the Austrian eID System to the Public Cloud. In: Janczewski, L., Wolfe, H., Shenoi, S. (eds.) SEC. IFIP AICT, vol. 405, pp. 300–314. Springer (2013)