Group Signatures on Mobile Devices: Practical Experiences

Klaus Potzmader, Johannes Winter, Daniel Hein, Christian Hanser, Peter Teufl[1], and Liqun Chen[2]

[1] Institute for Applied Information Processing and Communications,
Inffeldgasse 16a,
8010 Graz,
Austria
klaus.potzmader@student.tugraz.at,
{johannes.winter,daniel.hein,chanser,peter.teufl}@iaik.tugraz.at
[2] Hewlett-Packard Laboratories,
Long Down Avenue, Stoke Gifford,
Bristol, BS34 8QZ,
United Kingdom
liqun.chen@hp.com

Abstract. Group signature schemes enable participants to sign on behalf of a group in an anonymous manner. The upcoming ISO20008-2 standard defines seven such schemes, which differ in terms of capabilities, used crypto systems and revocation approaches. Further information about practical considerations, such as runtime performance or implementation overhead is considered useful when deciding for a certain scheme. We present a Java framework that allows for a detailed comparison of the mechanisms, of which three are already implemented. For these implemented mechanisms, a detailed performance evaluation is shown for both a notebook and Android-based mobile devices. Furthermore, significant experiences during implementing and evaluating the schemes as well as crucial bottlenecks are pointed out. We remain in the flexible Java environment, without special platform-specific optimizations. Using precomputation, we already achieve acceptable online signing timings. Signing times are considered most important given proposed application scenarios.

Keywords: Group signatures, performance evaluation, mobile devices, list signature scheme, DAA, e-voting, attestation, e-payment, Android, Java

1 Introduction

Group signature schemes as first introduced by Chaum and van Heyst [17] allow participants to sign messages on behalf of a group in an anonymous manner. The signature scheme ensures to a verifier by cryptographic means that the signer is indeed a valid member of such a group, but does not reveal the specific signer's

identity. The anonymity resides in the verifier not knowing what exact member actually signed the document. The degree of anonymity thus strongly depends on the group size.

Group signatures have many interesting applications, such as e-voting [16], e-bidding [21], online payment [29] or anonymous attestation [9]. Many of the above use-cases would additionally benefit from mobile usage. Mobile usage requires efficient implementations on mobile devices, such as mobile phones. Even with the advent of computationally powerful smartphones, implementing group signature schemes with reasonable security and response times is still a challenge. Group signatures are based on cryptographic algorithms which are considerably more complex than schemes that use a single public key per entity. The problem is only exacerbated by the sheer number and variety of different group signature schemes that have been proposed. Often these schemes are based on different cryptographic approaches, such as RSA or elliptic curve cryptography (ECC). In an effort to alleviate the above problems, ISO/IEC have proposed the ISO20008-2 [24,25] standard, which is currently a *Draft International Standard (DIS)* and undergoing public review.

In this paper, we present an evaluation of three different group signature schemes. All three mechanisms are part of the ISO20008-2 standard [25]. The first mechanism is a scheme for e-voting, proposed by Canard et al. [16]. It is designed to enable anonymous ballots, whilst still being capable of detecting double-voting without de-anonymizing the voter. ECC-DAA, the second mechanism we analyze, was originally introduced by Chen et al. [20] and is designed for the upcoming Trusted Platform Module (TPM) 2.0[3] specification. Used in conjunction with a TPM, ECC-DAA allows users to prove a specific load-time software state to a remote entity, called attestation. The remote verifier is thereby not able to identify the user, but can be sure that the platform report originates from a valid TPM. The final mechanism, the scheme by Isshiki et al. [23] gears towards using group signatures for identity management. Labels which uniquely define a specific group member shall be masqueraded behind a credential that represents the group as a whole. Therefore, a verifier does not learn any specifics about an enquiring member, only whether it is a valid member of a group or not.

The goal of our evaluation is to determine the applicability of those schemes for use on mobile devices. To achieve this, we have implemented a Java based framework for evaluating group signature schemes. We have designed the framework to be flexible enough to encompass the different cryptographic approaches of the three mechanisms, while also providing the common cryptographic primitives used by some of the three mechanisms. We have chosen Java because of its wide use in mobile phones based on Google's Android[4] operating system. Furthermore, it allows for detailed comparisons without platform-specific optimizations, thus providing a generalized overview.

[3] http://www.trustedcomputinggroup.org/resources/trusted_platform_module_specifications_in_public_review

[4] http://www.android.com/

Our framework can be extended to implement more than the three mechanisms we compare in this paper. Additionally, we have organized the framework to abstract cryptographic primitives from existing implementations in the Java runtime. The reason for this abstraction is ability to replace cryptographic primitives with different implementations both in Java and C and thus to evaluate their efficiency. Finally, we make most of the framework publicly available under an open source license[5].

In addition to just evaluating the schemes against each other, we also consider aspects such as optimization for a specific use case. For example, all steps for ECC-DAA can be precomputed if the use case requires total anonymity.

The paper is organized as follows. In Section 2, the properties, components and revocation approaches of group signature schemes are discussed in general. Moreover, the implemented schemes are summarized and compared. Section 3 provides a detailed comparison regarding runtime and memory usage of the schemes, both on a notebook and on mobile devices. In Section 4, these results are put into the context of other evaluations by merging them in a table. A summary of the results and future work aspects are given in Section 5.

2 Background

In the following, group signature schemes are discussed in a broader view, incorporating the related concepts of so-called list signature schemes and attestation.

Group signature schemes strive to fulfill the properties *Soundness and Completeness*, *Anonymity*, *Unforgeability*, *Traceability*, *Coalition Resistance*, *Non-Frameability* and *Unlinkability* as defined by Chaum and van Heyst [17] and extended by Bellare et al. [5, 6]. Note that opening is generally considered a mandatory feature for group signature schemes and linking is usually undesirable [17]. However, scenarios such as electronic voting have shown uses for types of schemes where opening is less critical and conditional linking is desirable to detect double usage. Canard et al. label these schemata as *list signature schemes* [16]. Therefore, traceability and unlinkability are considered optional features in this case to incorporate both attestation and list signature schemes. List signature schemes additionally require that adversaries can at most produce one valid signature per linking indicator and corrupted member. Any additional signature would cause the adversary to be either detected or to reveal the identity of the corrupted member.

Scheme Processes Group signature schemes are comprised of the following individual processes:

- **Group Establishment** The group is initially set up. That is, its public key and the corresponding group membership issuing key are created. The creating instance, now holding the membership issuing key, is called *issuing authority*.

[5] Available at https://github.com/klapm/group-signature-scheme-eval

- **Joining** New members get added to the group. This is done by a joint computation of the applicant and the issuing authority in a way that the private key of the applicant remains secret. The issuing authority issues a membership credential to complete the join process.
- **Signing** Valid members are able to sign documents. Signatures depend on both the private key and the membership credential, but do not reveal any of them in a computationally feasible way.
- **Verification** Using the group public key, a verifier is able to tell whether a given signature was issued by a valid member of the group.
- **Revocation** Existing members are being excluded from the group. Revoked members can no longer sign on behalf of the group. This can be achieved in several ways, which are discussed below.
- **Opening** Optional, depending on whether the scheme supports opening. A separate authority, the so-called *opening authority* is installed, able to open signatures and thus reveal the specific identity of a signature's author.
- **Linking** Optional, only applicable if linking is supported and enabled. Given two signatures, any stakeholder is able to tell apart whether these signatures were created by the same author.

Revocation A challenging task for group signature schemes is to remove existing members without affecting the workings of the group as a whole. Several approaches appear throughout the literature [19, 24], all of which have advantages and disadvantages:

- **Private Key Revocation** A compromised private key of a group member is added to a list of no longer valid keys. Verifiers, given a signature, are able to determine whether the signature was created using such a key. If the revocation list stores both the private key and the associated member id, a revocation check might immediately reveal the identity of the otherwise anonymous signer if her key was revoked before. Depending on where this list is stored and who has access to it, revocation can either be *global*, that is affecting all verifiers or *local*, per verifier [9].
- **Blacklist Revocation** Blacklist revocation is typically a local revocation, in which a verifier stores a list of no longer valid signatures. Given a new signature, the verifier is then able to determine whether this new signature was created by either a blacklisted or still valid author [9].
- **Signature Revocation** Signature revocation has the same effect as blacklist revocation, but uses a different approach to achieve it. With signature revocation, revoked signatures are kept in a list as well. A verifier then requires additional proof from the signer, showing that she is not the one who created any signatures on that list to accept her signature. Naturally, this approach impacts the overall verification performance. Signature revocation can be a local or global revocation, depending on where the list is stored and who has access to it [10].
- **Credential Revocation** In this scenario, the membership credentials of revoked members are stored in a list. Signers might then be required to prove

that their credential is not on that list. This proof is typically done implicitly, such that non-revoked signers prove their credibility by being able to still produce valid signatures. This mechanism is also referred to as credential update or re-keying and usually involves an update of the public key and membership credentials. Existing signers who are supposed to keep their membership are notified with means to update their membership credential when another member is about to be dismissed. Since the credential of the leaving member gets outdated, she is no longer able to participate. Credential revocation is a global revocation. Note that credential update invalidates existing signatures, as existing signatures will no longer verify when using the newly created public key [8, 14].

The implemented schemes are briefly explained in the following subsections, and their relevant properties are summarized for comparison in a table below.

2.1 Canard et al.

The first implemented scheme is the list signature scheme proposed by Canard et al. [16]. We will refer to it using its authors as name. List signature schemes support linking signatures as long as they are *tagged* with the same value. Following the author's original description, a tag is typically a time frame in which no two signatures are allowed, e.g., a voting period. Linking two signatures can be done by every stakeholder, without the need for any secret, called *public detection* by the authors. If two signatures were created using different tags, linkability is computationally infeasible.

The author's proposed scheme is based on work by Ateniese et al. [2] and supports multiple revocation processes, namely private key revocation, both locally and globally, and blacklisting. The scheme's security is based on both the strong RSA and the decisional Diffie Hellman (DDH) assumptions (cf. Section 2 in [16]).

2.2 Direct Anonymous Attestation

Direct Anonymous Attestation (DAA) was originally introduced by Brickell et al. [9] as a mechanism to remotely authenticate a trusted platform in a privacy-preserving way. It is intended to be used in conjunction with a Trusted Platform Module[6] (TPM) and, therefore, strictly splits the signing party into a computationally powerful *assistant signer* and a less powerful *principal signer*, the smaller TPM chip. While it differs in its purpose, it is similar to the previously described list signature scheme when comparing at a property level. DAA also supports linking, given the signatures were crafted using the same *linking base*, a generalization of what has previously been called *tag*.

For this work, the pairing-based ECC-DAA variant, as proposed by Chen et al. [20], was implemented using Barreto-Naehrig [3] curves. It was designed for the Trusted Computing Group[7] (TCG) and is now included in the TPM

[6] http://www.trustedcomputinggroup.org/developers/trusted_platform_module
[7] http://www.trustedcomputinggroup.org

2.0 specification. As we do not utilize a TPM in this setting, the strict split-up between assistant and principal signer was omitted and both components are merged into one signing party.

The ECC-DAA variant is most flexible in terms of revocation, supporting private key revocation, signature revocation, both either locally or globally, blacklisting and credential update. It is provably secure under the DDH, Lysyanskaya-Rivest-Sahai-Wolf (LRSW) and static Diffie-Hellman (SDH) assumptions, see Section 1 in [20].

2.3 Isshiki et al.

The third implemented scheme was introduced by Isshiki et al. [23] and is an adaption of the Camenisch-Groth scheme [13], enabling faster revocation. The scheme is RSA-based and supports opening, with the opening capability built on top of an additional elliptic curve group.

In this scheme, the opening authority is entirely separated from the issuing authority. The issuing authority holds the secret membership issuing key, which corresponds to the group's public key. The opening authority holds another secret, called *group membership opening key*. Its counterpart is the so-called *group opening key* and yet another public key for the group. To achieve this setup, both authorities are involved during the initial group establishment. The scheme supports credential update as revocation mechanism and is secure under the strong RSA and DDH assumptions, see Section 4.2 in [23].

2.4 Comparison Summary

The properties of the three implemented schemes are summarized in Table 1. The reason for choosing these three schemes is that they vary in both their goals and their construction. The intention is to gain a principal overview of the three schemes, and measurements that are roughly mappable to related schemes based on similar principles. For example, schemes such as the one's described in [2,13].

Scheme	Optional Linkability	Openable	Crypto-System(s)	Intractability Assumptions	Revocation Support CU	PKR	BL	SR
Canard et al.	✓		RSA	Strong RSA, DDH		✓	✓	
ECC-DAA	✓		ECC, Pairings	DDH, LRSW, SDH	✓	✓	✓	✓
Isshiki et al.		✓	RSA, ECC	Strong RSA, DDH	✓			

CU: Credential Update, PKR: Private Key Revocation, BL: Blacklisting, SR: Signature Revocation

Table 1. Scheme comparison

3 Results and Evaluation

Performance measurements regarding both runtime and memory were gathered on an off-the-shelf notebook as well as on multiple mobile devices. Concerning the mobile scenario, especially signing is critical. Signing and verifying denote regular tasks, whereas group creation and joining are sporadic, if not one-time events. In mobile scenarios, these two operations have to perform sufficiently fast in time-constrained settings, for example when placing an order using the mobile phone. Therefore, the presented analysis concentrates on these two operations.

Before presenting concrete measurements, the framework and test setup are described. Furthermore, we emphasize that we conducted all tests without active revocation, even though the framework supports it. After discussing the actual results, we conclude this section with the key lessons we learned.

3.1 Framework

All evaluated group signature schemes are embedded in a common framework and use the same implementation of cryptographic primitives. The framework's purpose is to provide a flexible environment for performance measurement and future extension, but it is limited to the task of comparing individual schemes. Therefore, we did not implement standalone components and message passing is done entirely locally by simple method invocations. Hence, the measurements do not include network communication overhead.

The framework is written in pure Java and does not rely on any external libraries. The same code base runs on both Java Standard Edition and Android without requiring any special adjustments. The group signature schemes are unified under a common interface and the surrounding evaluation code is the same for all concrete implementations. The implementation supports all necessary operations, including revocation mechanisms, but focus was put on measurements for signing and verification. As a consequence, sub-protocols such as proving the knowledge of a discrete logarithm in the scheme of Isshiki et al. were omitted.

The pairing map, as required by ECC-DAA, is essentially a Java port of the Optimal Ate Pairing C implementation[8] provided by Beuchat et al. [4]. We ported the version of January 2013 from Beuchat et al., which in turn benefited from insights gained by Aranha et al. [1]. Porting from an assembler-optimized C implementation to an interpreted language, such as Java, has various drawbacks regarding runtime performance. Some of the optimizations have to be abandoned and choosing Java comes with further performance impacts, such as just-in-time compilation and garbage collection. These circumstances leave us at considerably slower timings of about 7ms for a single pairing evaluation, compared to about 0.5ms of the original C version, measured on the same notebook. Nevertheless, Java was chosen to allow for general comparability of the schemes by disregarding platform-specific optimizations and to allow easy portability.

[8] Available at http://homepage1.nifty.com/herumi/crypt/ate-pairing.html

3.2 Test Setup

The notebook case tests were performed on a Lenovo ThinkPad T420s notebook, equipped with an Intel i7-2620M CPU at 2x2.7GHz and 8GB of RAM using Windows 8 x64 and Java 1.7.0_11, 64 bit. For the mobile case, different devices were used during the evaluation. The devices and their relevant specifications are listed in Table 2.

Device	Galaxy Nexus i9250	Galaxy S Plus i9001	Galaxy S3 GT-I9300
Manufacturer	Samsung	Samsung	Samsung
Operating System	Android 4.2	Android 2.3.6	Android 4.0.4
CPU	2x1.2GHz ARM Cortex-A9	1.4GHz ARM Cortex-A9	4x1.4GHz ARM Cortex-A9
System-on-Chip	Texas Instruments OMAP 4460	Qualcomm Snapdragon S2 MSM8255	Samsung Exynos 4412
Memory	1024MB	1024MB	1024MB

Table 2. Devices used for the evaluation

The shown evaluations are not always directly comparable due to the different crypto-systems. In fact, a key length of 256 bit in ECC-DAA implies 128 bit security strength. The schemes by Canard et al. and Isshiki et al. evaluate to 112 bit security strength at a parameterization with a modulus length of 2048 bit. *Security strength* in this context denotes the number of operations required to break a cryptographic algorithm, specified in bits; so 80 bit security strength means at least 2^{80} required operations. Table 3 summarizes the evaluated parameterizations, of which all except the 80 bit setup of Canard et al. are recommended choices by the ISO20008-2.2 draft standard. Note that the value l_p at the scheme by Canard et al. refers to a single factor of the composite modulus. For details on these parameters, we refer the reader to the ISO20008-2 draft [25] or the original sources [16, 20, 23]. The following results will refer to these setups using the modulus length as indicator.

All measurements were conducted without a revocation mechanism in place. Enabling revocation when evaluating the schemes adds numerous additional parameters, such as the group size or the specific revocation approach used. The different revocation approaches are hardly comparable, with strong dependencies on either the group size or the number of already-revoked members. Furthermore, the introduced revocation approaches are not always influencing the verification time. For example, signature revocation might influence signing, and credential update is an entirely separate process. Considering these differences, we decided to exclude revocation from the evaluation, despite its impact for practical purposes. However, the framework supports revocation, so revocation experiments can be carried out as well.

Scheme	Canard et al.		ECC-DAA	Isshiki et al.	
Parameterization	$l_p = 512$	$l_p = 1024$	$t = 256$	$K_n = 1024$	$K_n = 2048$
	$k = 160$	$k = 160$	$p = \|256\|$	$K = 160$	$K = 224$
	$l_x = 160$	$l_x = 160$		$K_c = 160$	$K_c = 224$
	$l_e = 170$	$l_e = 170$		$K_s = 60$	$K_s = 112$
	$l_E = 420$	$l_E = 420$		$K_e = 504$	$K_e = 736$
	$l_X = 410$	$l_X = 410$		$K'_e = 60$	$K'_e = 60$
	$\epsilon = 5/4$	$\epsilon = 5/4$			
Security Strength	80 bit	112 bit	128 bit	80 bit	112 bit

Table 3. Scheme security comparison

3.3 Runtime

The following data was gathered by averaging the single processes over 100 iterations. The mean values are explicitely shown, whereas the standard deviation is only indicated using error bars. Signing was computed using a different message per sign operation, though within the same group. Verification was also varied in terms of message and signature.

Runtimes are split into the notebook and mobile cases. The mobile case is discussed in more detail, including precomputation. Parts of the signature, which do not depend on the message can be precomputed, such that the online signing time is further decreased. Additionally, if optional linkability is disabled in ECC-DAA and in the scheme by Canard et al., then most of the signature attributes are precomputable. We will leverage this ability by shifting workload to a precomputation phase.

Furthermore, we measured the runtimes with two different primitive arithmetic implementations for elliptic curve cryptography, namely the default `java.math.BigInteger`, denoted *BigInteger* from now on, and a custom implementation, denoted *custom*. There were particular reasons for choosing another underlying implementation, which will be discussed later on.

Notebook Runtimes for signing and verifying on a notebook-like environment are given in Figure 1. We can see that the parameter length and, thus, the length of the internally used values roughly dictates the runtimes. In this setting, ECC-DAA outperforms the schemes by Canard et al. and Isshiki et al. It is faster in both signing and verifying and provides higher security strength at the same time. Indeed, all schemes are fast enough to be used out of the box, with no measurement exceeding 335ms. There are only small differences between the two used arithmetic implementations, with the custom one being marginally slower.

Mobile Device On Android, a different picture emerges. The same implementation, deployed as an app, results in the signing times as depicted in Figure 2. We can see that there is not just simple upscaling in place, resulting from the

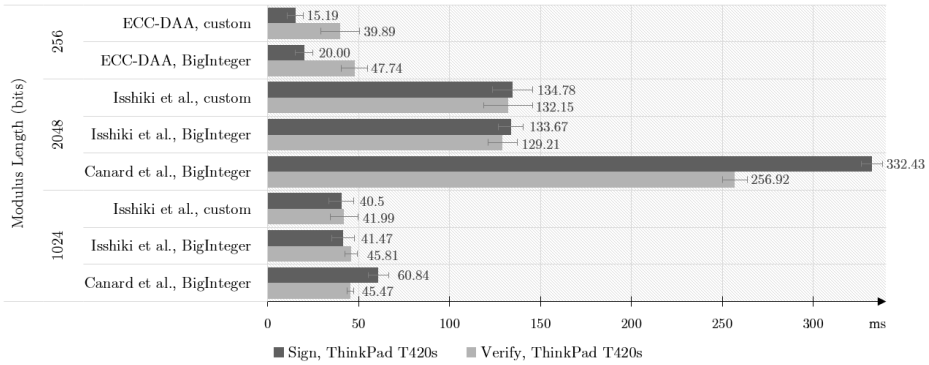

Fig. 1. Signing and verifying, notebook case

limited computational power of the devices. Contrary to the notebook case, the non-ECC scheme by Canard et al. does no longer exceed the runtimes of the other two schemes that much. Furthermore, we see stronger differences between using Java's default BigInteger and the custom implementation.

Before discussing the reasons of these runtime effects, we show the achieved online times when utilizing the fact that certain parts of the signatures are precomputable.

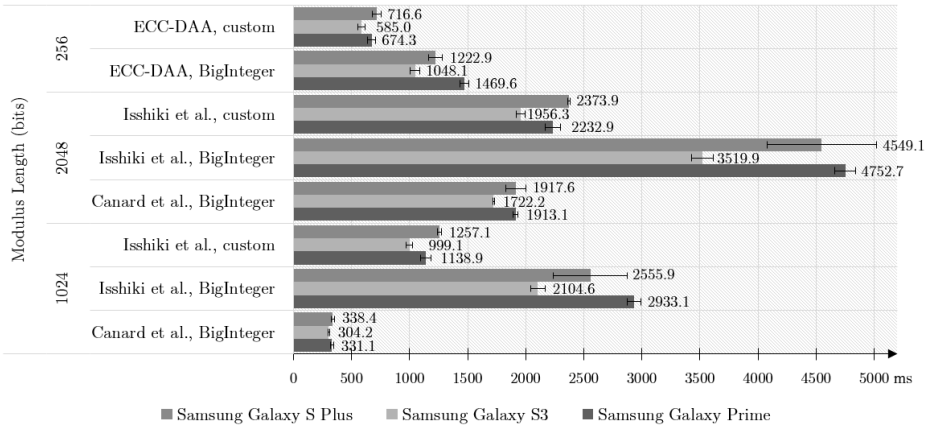

Fig. 2. Signing without precomputation, mobile case

Precomputation ECC-DAA and the scheme by Canard et al. support optional linkability by computing a linking base into the signature, given that linkability is desired and the verifier shall be able to provide the linking-base at sign time. In this scenario, only those parts of the signature not depending on either the

message or the linking base can be precomputed. We refer to this case using the term *partial precomputation*.

If linkability is not required in a certain use case, it can be disabled by setting the linking base to a constant value. Here, almost the whole signature is precomputable, thus vastly decreasing the online signing time. This will be called *full precomputation* from now on.

The scheme by Isshiki et al. does not support linking, but allows for extensive precomputation as well, referred to as plain *precomputation*. Figure 3 is a comparison of the evaluated schemes using partial (PPC) as well as full (FPC) precomputation for ECC-DAA and the scheme by Canard et al., and precomputation (PC) for the scheme by Isshiki et al.

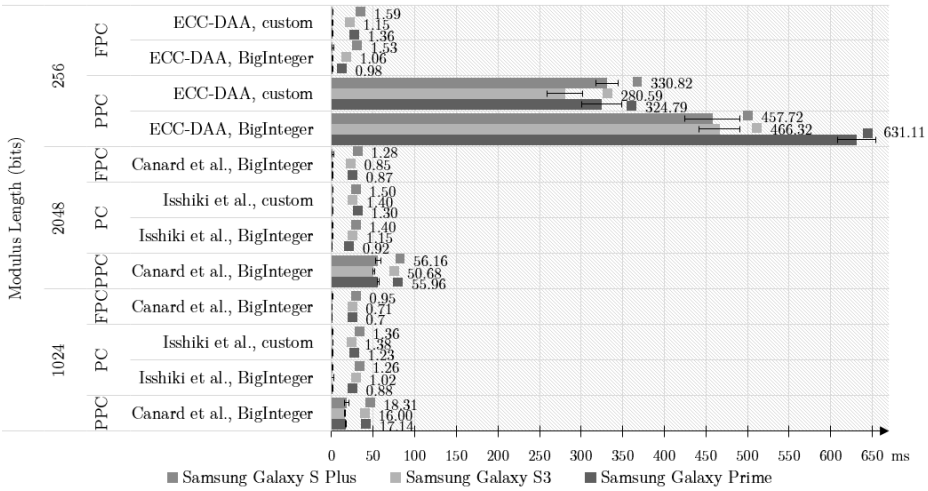

Fig. 3. Signing with precomputation, mobile case

ECC-DAA with partial precomputation leaves three point multiplications for the online signing phase, whereas seven out of seven can be precomputed in case linkability is not needed. The gap between precomputations is smaller for the scheme by Canard et al., where the difference is essentially one hash computation and three modular exponentiations. Precomputing the signature for Isshiki et al. reduces the needed operations at signing time to one hash computation as well as five additions, multiplications and reductions. Given the differing complexity in the remaining operations, the shown gap in runtime is reasonable. With full precomputation, the ECC-DAA workload is reduced to a single hash computation as well as one plain additon, multiplication and reduction.

Verification As depicted in Figure 4, the notebook results are almost inverted. As in the mobile signing setting, ECC-DAA and the scheme by Isshiki et al. are considerably slower than the scheme by Canard et al. Again, this partially stems

from implementation issues when using elliptic curve cryptography. However, verification is a less critical factor regarding mobile device performance when considering typical application scenarios for group signature schemes. Commonly, verification can take place at more powerful devices [9,16,21,30].

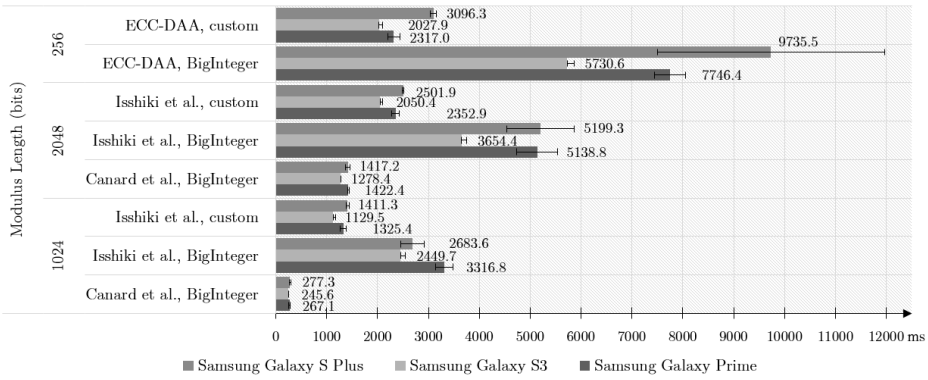

Fig. 4. Verification, mobile case

3.4 Lessons Learned

The runtime anomalies that emerge when comparing the notebook and mobile cases are partially implementation-related issues. The elliptic curve cryptography part of the framework, used by ECC-DAA and the scheme by Isshiki et al., is comparably slower on mobile devices. The dominating factor for this is the amount of temporary objects that are instantiated to hold an intermediate value for a short amount of time and are then of no use anymore. Thus, a lot of runtime and memory is wasted on instantiating, copying and then garbage collecting these objects.

The main reason for this is Java's immutable `BigInteger` implementation. Each operation on a BigInteger leads to the allocation of a new object, storing the result. Considering that BigIntegers are the most basic element, used by all prime and extension field operations, this has a major impact on runtime. These temporary instances are less of a problem with operations, such as modular exponentiation, as used by the schemes by Canard et al. and Isshiki et al. Modular exponentiation is provided by Java's BigInteger and uses a mutable variant internally. The problem becomes apparent for operations built on top of BigIntegers, such as point multiplication or pairings. Therefore, the scheme by Canard et al. was only tested with BigIntegers, as the expected gain from using the custom implementation is very low.

While this is generally a non-issue in the notebook case, it has noticeable impact when the algorithms are run on an Android based mobile device. The

reason for this is much slower garbage collection in combination with a strict collection enforcement policy.

The garbage collection overhead is severe enough that point multiplication is faster using affine coordinates than a mix of Jacobian and affine coordinates, to give an example. Using mixed coordinates in point multiplication allows to omit expensive modular inversions during point addition and doubling within the double-and-add cycle and is thus usually the faster variant [12,30]. However, using Java's BigInteger, point multiplication in affine coordinates turned out to be faster, since it requires less intermediate instances.

Android garbage collection is triggered as soon as there are enough collectable objects, regardless of the overall free heap. Therefore, the small overall memory footprint of our implementation poses no advantage in that regard. Originally, our implementation was developed to give an overview of the different schemes. It turned out that there is a strong indication that such a portable implementation is almost powerful enough to be used already, without the need to resort to highly optimized, but device dependent implementations. In light of this and also to decouple this platform dependent implementation factor, we aimed at alleviating the problem by reducing the amount of intermediate instances.

We decided to use a custom integer arithmetic implementation and plug it underneath the field operations. Since this custom implementation is designed to enforce in-place operations wherever possible, lots of instantiations are spared. Furthermore, fixed-width integer arithmetic is used, allowing for easier recycling of no longer used objects. Taming the garbage collector this way reduces runtimes dramatically. The problem is ameliorated, but there are still a few garbage collection runs that cost considerable runtime. Nevertheless, the attempt to decrease runtimes is successful, considering the visible runtime drops in the measurements.

3.5 Memory

The exact amount of memory consumed by the schemes alone is difficult to pinpoint in a managed, garbage-collected environment such as Java. On the notebook, we approximated a heap consumption of below 5.5 megabytes and a permanent generation space of about 4MB. Lower artificially introduced memory limits were just ignored by the JVM. Therefore, these values are a loose upper bar. As we see from experiments on the mobile devices themselves, the schemes run with less than that. For example, on the Samsung Galaxy S3 device we used, the initial free heap memory per app is as low as one megabyte, since the remaining heap is already filled with preloaded Android elements. The implementation ran under these circumstances without further heap growth, thus having only one megabyte of heap available seems to be enough.

Code Size The code size was determined by the resulting size of the `classes.dex` file when exporting an APK package. ProGuard[9] was enabled when doing

[9] http://developer.android.com/tools/help/proguard.html

these tests, which reduces the size by renaming variable names to commonly short identifiers. Table 4 lists the code size for each module.

Complete Set	Canard et al.	ECC-DAA	Isshiki et al.	Framework
349KB	163KB	249KB	257KB	273KB

Table 4. Code size comparison

Complete set refers to the whole functionality, including ECC/Pairing code and all schemes. *Framework* on the other hand is infrastructure material only. The separate scheme sizes denote the code size of both the specified scheme itself and the required framework code for it to work, hence the overlaps.

Since the scheme by Canard et al. does not use any elliptic curve cryptography, its code size is naturally smaller, as the ECC part of the framework can be dropped. More than half of the overall code size is consumed by underlying primitive functionality. ECC-DAA and the scheme by Isshiki et al. are almost equal in size. Generally, code size is low enough to be run on mobile devices with the whole package having a size of less than 350KB.

4 Related Work

Group signatures have become an extensive field of research with many different proposed schemes and suggested approaches. For surveys on group signature schemes themselves, we refer the reader to [26,31].

Implementational aspects of group signatures were evaluated in various related publications. In Table 5, several publications are put into context to allow for a rough overview of current advances. Unfortunately, a direct comparison is not possible, since all approaches differ in terms of the test setup, used cryptosystem and the scheme itself. Furthermore, vital details such as the used pairing map are omitted as well. The intention is to give a brief overview on how a Java-based implementation fits into this picture, both in the mobile and the notebook case. Still, these values have to be taken with a grain of salt, especially when comparing individual results. Most shown implementations are C-based and use the pairing-based cryptography library PBC[10]. A high-level implementation in Java typically cannot compete with low-level C implementations using partial assembler optimizations, not to mention hardware solutions based on Application-Specific Integrated Circuits (ASICs) or Field-Programmable Gate Arrays (FPGAs). However, the enhanced flexibility of platform indepence might be worth the longer runtimes. The additional runtimes seem acceptable, especially when comparing our results with the native code estimations from Manulis et al. [26].

[10] See https://crypto.stanford.edu/pbc/

	Source	Remarks	Scheme	Setup	Strength	Sign	Verify
Hardware	Morioka et al. [27]		Isshiki et al. [23]	ASIC	80 bit	135ms	135ms
	Manulis et al. [26]	C/ASM, PBC, estimations	Boneh-Shacham [8]	FPGA	128 bit	128.7ms	144.76ms
			Bichsel et al. [7]	FPGA	128 bit	58.3ms	52.3ms
Mixed	Chen, Page, Smart [20]	C/ASM	RSA-DAA [9]	TPM	128 bit	33700ms	194ms
			ECC-DAA [20]			3400ms	48.31ms
	Canard et al. [15]	C, PBC	XSGS [22], online signing	WSN	80 bit	< 200ms	55ms
			XSGS [22], offline signing		80 bit	3995-6596ms	55ms
Smartphone	Manulis et al. [26]	Android NDK, PBC, estimations	Camenisch-Groth [13]	PHONE	128 bit	431.5ms	431.5ms
			Boneh-Shacham [8]			1211.7ms	2118ms
			Bichsel et al. [7]			1002.3ms	1577.4ms
PC	Manulis et al. [26]	C/ASM, PBC, estimations	Camenisch-Groth [13]	MPC	128 bit	170.4ms	170.4ms
			Boneh-Shacham [8]	MPC		402.4ms	691.6ms
				MPC*		131.4ms	150.1ms
			Bichsel et al. [7]	MPC		332ms	508ms
				MPC*		61ms	56.8ms
	Bringer and Patey [11]	C++	Boneh-Shacham [8]	BPPC	80 bit	1000ms	1170ms
			Chen-Li (patched) [11,18]			450ms	400ms

Legend:

ASIC	0.25um ASIC Implementation, 100MHz clock frequency, by Morioka et al. [27]
BPPC	Intel Core2Duo, 2.93GHz
FPGA	Xilinx Virtex-6 FPGA prototype, assumptions based on [32]
MPC	Intel Core Duo LV L2400, 1.66GHz
MPC*	Same as MPC, but assuming the pairing measurements as shown by Naehrig et al. [28]
PHONE	HTC Desire, 1 GHz Qualcomm ARMv7 Scorpion CPU
TPM	33MHz ARMv7, emulated TPM; 2.4 GHz Intel Core2-6600 host platform
WSN	MICAz/TelosB wireless sensor node platforms; 1.4 GHz Intel Core2Duo CPU host platform, see [15]

Table 5. Comparison of other published results

5 Conclusion and Future Work

Implementing group signature schemes in plain Java for the Android operating system yielded various interesting insights. Group signature schemes are considered ready for scenarios, where signature creation is performed on mobile devices and verification is done on a more powerful device. The shown schemes by Canard et al., Chen et al. and Isshiki et al. allow for precomputing parts of the signature, enabling tolerable online signature timings. The concrete timings are differing between the schemes and use cases, but are generally considered acceptable. The worst case scenario runtime we measured averages at about 330ms for online signing, with a huge gap to the second longest run of about 56ms. Newer smartphone generations seem to improve these results significantly. Verification runtimes are in part noticeably slower, but common scenarios only require signing to be performed on low-power end user devices. Therefore, we consider the technology ready to be used on mobile devices. A partial native code implementation might even allow for omitting the precomputation step whilst still delivering acceptable runtimes.

The framework did not only enable detailed comparisons of the schemes, it also revealed factors of considerable impact regarding the runtime environment. The Android garbage collector turned out to be the main bottleneck. Therefore, having less intermediates and recycling instances is of great importance on Android. The runtimes of individual operations tend to be tightly coupled with the amount of required intermediate instances. Cumulative effects, such as the garbage collector kicking in after a certain threshold of objects to collect might also hinder runtime estimations based on the timings of individual operations.

The accompanying framework is open source and allows further extension with the remaining schemes defined in ISO20008-2 [25] or other, similar schemes.

Subsequent anticipated steps are to loosen the platform independence a bit and to implement parts of the framework using the Android Native Development Kit. Given the unmanaged memory environment, this is expected to result in further acceleration.

References

1. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In: Peterson, K. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (2011)
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In: Bellare, M. (ed.) Crypto 2000. LNCS, vol. 1880, pp. 11–15. Springer, Heidelberg (2000)
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
4. Beuchat J.-L., González-Díaz J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-Speed Software Implementation of the Optimal Ate Pairing over BarretoNaehrig Curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp, 21–39. Springer, Heidelberg (2010)

5. Bellare, M., Micciancio, D., Warinischi, B.: Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In: Biham, E., (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003)
6. Bellare, M., Shi, H., Zhang, C.: Foundations of Group Signatures: The Case of Dynamic Groups. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (2005)
7. Bichsel, P., Camenisch, J., Neven, G., Smart, N.P., Warinischi, B.: Get Shorty via Group Signatures without Encryption. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 381–398. Springer, Heidelberg (2010)
8. Boneh, D., Shacham, H.: Group Signatures with Verifier-local Revocation. In: 11th ACM Conference on Computer and Communications Security, pp. 168–177. ACM Press, New York (2004)
9. Brickell, E., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: 11th ACM Conference on Computer and Communications Security, pp. 132–145. ACM Press, New York (2004)
10. Brickell, E., Li, J.: Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. In: 6th ACM Workshop on Privacy in the Electronic Society, pp. 21–30. ACM Press, New York (2007)
11. Bringer, J., Patey, A.: Backward Unlinkability for a VLR Group Signature Scheme with Efficient Revocation Check. In: Cryptology ePrint Archive, Report 2011/376. http://eprint.iacr.org/2011/376 (2011)
12. Brown, M., Hankerson, D., López, J., Menezes, A.: Software Implementation of the NIST Elliptic Curves over Prime Fields. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 250–265. Springer, Heidelberg (2001)
13. Camenisch, J., Groth, J.: Group Signatures: Better Efficiency and New Theoretical Aspects. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 120–133. Springer, Heidelberg (2005)
14. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
15. Canard, S., Coisel, I., De Meulenaer, G., Pereira, O.: Group Signatures are Suitable for Constrained Devices. In: Rhee, K.-H., Nyang, D. (eds.) ICISC 2010. LNCS, vol. 6829, pp. 133–150. Springer, Heidelberg (2010)
16. Canard, S., Schoenmakers, B., Stam, M., Traoré, J.: List Signature Schemes. J. Discrete Applied Mathematics 154 (2), 189–201 (2006)
17. Chaum, D., van Heyst, E.: Group Signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)
18. Chen, L., Li, J.: VLR Group Signatures with Indisputable Exculpability and Efficient Revocation. In: 2nd IEEE International Conference on Social Computing, pp. 727–734. IEEE Press, New York (2010)
19. Chen, L., Li, J.: Revocation of Direct Anonymous Attestation. In: Chen, L., Yung, M. (eds.) INTRUST 2010. LNCS, vol. 6802, pp. 128–147. Springer, Heidelberg (2011)
20. Chen, L., Page, D., Smart, N.P.: On the Design and Implementation of an Efficient DAA Scheme. In: Gollmann, D., Lanet, J-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 223–237. Springer, Heidelberg (2010)
21. Chen, L., Pedersen, T.P.: New group signature schemes. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 171–181. Springer, Heidelberg (1995)

22. Delerablée C., Pointcheval, D.: Dynamic Fully Anonymous Short Group Signatures. In: Nguyen, P. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 193–210. Springer, Heidelberg (2006)
23. Isshiki, T., Mori, K., Sako, K., Teranishi, I., Yonezawa, S.: Using Group Signatures for Identity Management and its Implementation. In: 2nd ACM workshop on Digital Identity Management, pp. 73–78. ACM Press, New York (2006)
24. ISO/IEC 20008-1: Information technology - Security techniques - Anonymous digital signatures - Part 1: General. Stage 40.20. International Organization for Standardization. Geneva, Switzerland. (2012)
25. ISO/IEC 20008-2: Information technology - Security techniques - Anonymous digital signatures - Part 2: Mechanisms using a group public key. Stage 40.20. International Organization for Standardization. Geneva, Switzerland. (2012)
26. Manulis, M., Fleischhacker, N., Günther, F., Kiefer, F., Poettering, B.: Group Signatures - Authentication with Privacy, a study issued by the German Federal Office for Information Security (BSI). https://www.bsi.bund.de/ContentBSI/Publikationen/Studien/GroupSignatures/GruPA.html (2012)
27. Morioka, S., Isshiki, T., Obana, S., Nakamura, Y., Sako, K.: Flexible Architecture Optimization and ASIC Implementation of Group Signature Algorithm using a Customized HLS Methodology. In: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 57–62. IEEE Press, New York (2011)
28. Naehrig M., Niederhagen, R., Schwabe, P.: New Software Speed Records for Cryptographic Pairings. In: Abdalla, M., Barreto, P.L.S.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 109–123. Springer, Heidelberg (2010)
29. Popescu, C.: An Electronic Cash System Based on Group Blind Signatures. J. Informatica 17 (4), 551–564 (2006)
30. Rivain, M.: Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves. In: Cryptology ePrint Archive, Report 2011/338. http://eprint.iacr.org/2011/338 (2011)
31. Wang, G.: Security Analysis of Several Group Signature Schemes. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 252–265. Springer, Heidelberg (2003)
32. Yao G. X., Junfeng, F., Cheung, R. C. C., Verbauwhede, I.: A High Speed Pairing Coprocessor Using RNS and Lazy Reduction. In: Cryptology ePrint Archive, Report 2011/258. http://eprint.iacr.org/2011/258 (2011)