Performance Evaluation of Primitives for Privacy-Enhancing Cryptography on Current Smart-Cards and Smart-Phones

Jan Hajny[1(✉)], Lukas Malina[1], Zdenek Martinasek[1], and Ondrej Tethal[2]

[1] Cryptology Research Group, Department of Telecommunications, Brno University of Technology, Brno, Czech Republic
{hajny,crypto}@feec.vutbr.cz
http://crypto.utko.feec.vutbr.cz
[2] OKsystem, Prague, Czech Republic
tethal@oksystem.cz

Abstract. The paper deals with the implementation and benchmarking of cryptographic primitives on contemporary smart-cards and smart-phones. The goal of the paper is to analyze the demands of today's common theoretical cryptographic constructions used in privacy-enhancing schemes and to find out whether they can be practically implemented on off-the-shelf hardware. We evaluate the performance of all major platforms of programmable smart-cards (JavaCards, .NET cards and MultOS cards) and three reference Android devices (a tablet and two smart-phones). The fundamental cryptographic primitives frequently used in advanced cryptographic constructions, such as user-centric attribute-based protocols and anonymous credential systems, are evaluated. In addition, we show how our results can be used for the estimation of the performance of existing and future cryptographic protocols. Therefore, we provide not only benchmarks of all modern programmable smart-card platforms but also a tool for the performance estimation of privacy-enhancing schemes which are based on popular zero-knowledge proof of knowledge protocols.

Keywords: Cryptography · Privacy · Benchmark · Primitives · Proof of knowledge protocols · Smart-cards · Smart-phones

1 Introduction

With the increasing number and complexity of electronic services and transactions, the role of cryptography becomes more and more important. While the classical cryptographic algorithms for ensuring data confidentiality, authenticity and integrity are mostly well analyzed [1,2], the modern cryptographic primitives which are used as the building blocks of many advanced privacy-enhancing schemes remain theoretical and without any evaluation on real-world devices. Therefore, the goal of this paper is to identify the most frequent cryptographic primitives, which are being used in, e.g., digital identity protection

J. Garcia-Alfaro et al. (Eds.): DPM 2013 and SETOP 2013, LNCS 8247, pp. 17–33, 2014.
DOI: 10.1007/978-3-642-54568-9_2,

schemes, attribute-based authentication schemes and credential schemes, and analyze these primitives on commercially available devices.

For our benchmarks, we chose mobile and personal devices. The reason is that these devices are becoming the most popular ones for personal electronic transactions. For user authentication, eIDs and access control, the smart-cards are already the most preferred devices. For Internet transactions, the mobile phones and tablets are becoming the best choice for many users today. Thus, we chose smart-cards, mobile phones and tablets as the platforms for our benchmarks. We ran the benchmarks on all major programmable smart-card platforms, namely on JavaCards [3], .NET cards [4] and MultOS cards [5]. We chose Android as the platform for the smart-phone and tablet benchmarks because Android, together with Apple iOS, is the preferred operating system for mobile devices worldwide [6].

1.1 Related Work

We consider classical cryptographic constructions, such as RSA signatures, DSA signatures, hashes and symmetric block ciphers, to be well analyzed according to their speed on low-performance devices. A complex analysis of modern symmetric encryption algorithms is provided in [1]. Here, a selection of 12 block ciphers is evaluated on 8-bit microcontrollers. Furthermore, the paper [2] deals with the benchmarking of modern hash functions. A selection of 15 hashes (some of them in more versions) was evaluated on 8-bit microcontrollers. These rigorous benchmarks can be taken as a rich source of information when implementing the classical cryptographic systems.

On the other hand, there is a lack of information when someone needs to implement advanced privacy-enhancing schemes which employ provably secure protocols such as Σ-protocols [7], proof of knowledge (PK) protocols [8] or cryptographic commitments [9]. In fact, there are many theoretical cryptographic schemes which use these constructions without any analysis of implementability. The most well-known examples are group signature schemes [10], verifiable encryption schemes [11], anonymous credential systems [12] and Vehicular Ad-hoc Networks (VANETs). Unfortunately, only little information about the performance of these protocols can be inferred from the implementation papers [14,15]. These papers usually provide information about the overall performance of the schemes, but little about the performance of the building blocks used. Other papers [13,16] present only partial information. Since the building blocks are usually shared among many privacy-enhancing schemes, the information about their performance would be very useful for the evaluation of many unimplemented schemes and newly emerging theoretical constructions.

1.2 Our Contribution

In this paper, we provide benchmarks of selected cryptographic primitives on all major smart-card platforms and three Android devices. We chose the cryptographic primitives which are commonly used in modern privacy-enhancing

schemes and which have not been evaluated on resource-limited devices yet. These primitives are described in Sect. 2. The testing environment is described in Sect. 3. The actual benchmarks are included in Sect. 4. Finally, short analysis of results and the examples of how to use our results for the performance estimation of novel schemes is provided in Sects. 4.3 and 4.4.

2 Cryptographic Constructions

We briefly introduce the cryptographic constructions selected for benchmarking in this section. We chose the constructions and protocols which are often used in privacy-enhancing schemes. These constructions work as the building blocks and are modularly used in many today's schemes (such as IBM's Idemix [17], Microsoft's U-Prove [18], HM12 [19], etc.). These and similar schemes are further used in many privacy-enhancing applications (such as access control services, inter-vehicular communication, electronic IDs, e-cash) whose description is out of scope of this paper.

2.1 Classical Algorithms

We call well-known cryptographic algorithms, such as block ciphers, digital signatures and hash functions, the classical algorithms. These algorithms are usually provided directly by the API (Application Programming Interface) of almost all smart-cards and smart-phones. The examples of most common classical algorithms are DES [20], 3DES, AES [21] block ciphers and RSA [22], DSA [23] digital signatures and MD5 [24], SHA-1, SHA-2 [25] hash functions.

2.2 Commitment Schemes

A cryptographic commitment scheme can be used in scenarios where a user is required to bind to a number (e.g., a secret key) without disclosing it. There are two properties which must be fulfilled. They are the *hiding property* and the *binding property*.

- **Hiding property**: it is difficult[1] to learn the secret number from the knowledge of the commitment.
- **Binding property**: once committed to a number, the user cannot change it without changing the commitment.

Discrete Logarithm Commitment Scheme. Mostly, the DL commitment scheme works with the subgroup $\mathbb{Z}_q^*$ of a multiplicative group $\mathbb{Z}_p^*$. The subgroup $\mathbb{Z}_q^*$ is defined by a generator g of order q in $\bmod p$, where q and p are large primes and q divides $p-1$. The same group is used in the Digital Signature Algorithm (DSA) [23]. Numbers g, q, p are system parameters which are made public. To commit to a number $w < q$, a user computes $c = g^w \bmod p$. The user can later decide to open the commitment by making w public.

[1] It is either impossible or computationally unfeasible.

Pedersen Commitment Scheme. The systems parameters g, q, p, used in the DL commitment scheme, can be also used in the Pedersen scheme [9]. Additionally, one more generator h is used. It is important that $\log_g h \bmod p$ is unknown to the user. The commitment to a secret number w is computed as $c = g^w h^r \bmod p$ where r is a random number smaller than q chosen by the user. The user can later decide to open the commitment by making (w, r) public.

2.3 Proof of Knowledge of Discrete Logarithm Protocols

The proof of knowledge (PK) protocols can be used by a Prover to give a proof of knowledge of discrete logarithm ($PKDL$). Using the proof of knowledge protocol, the Prover is able to convince a Verifier that he knows $w = \log_g c \bmod p$ from the aforementioned DL commitment without actually disclosing the secret value w. In the CS notation [8], which we use throughout the paper, the protocol is denoted as $PK\{w : c = g^w \bmod p\}$. The most used practical $PKDL$ protocol for general groups, called Schnorr protocol [26], is recalled in Fig. 15 in Appendix.

2.4 Proof of Discrete Logarithm Equivalence

Using the proof of knowledge protocols, it is easy to give a proof that two different DL commitments c_1, c_2 were constructed using the same exponent w, so that $w = \log_{g_1} c_1 = \log_{g_2} c_2 \bmod p$. For this type of proof, the proof of discrete logarithm equivalence ($PDLE$) protocols can be used. The protocol is then denoted as $PK\{w : c_1 = g_1^w \bmod p \wedge c_2 = g_2^w \bmod p\}$. A practical example based on the Schnorr protocol is recalled in Fig. 16 in Appendix.

2.5 Signatures and Other Derived *PK* Protocols

In the last three sections, we introduced simple cryptographic primitives which are very often used as the building blocks in more advanced schemes. In our examples, we described very simple protocols only. Nevertheless, these protocols can be modularly combined in much more complex systems. For example, the proof of knowledge protocols can be adapted to the proofs of knowledge of DL representation of a public value c with respect to multiple generators $g_1, g_2, \ldots, g_i$. Such a protocol is described in CS notation as $PK\{(w_1, w_2, \ldots, w_i) : c = g_1^{w_1} g_2^{w_2} \ldots g_i^{w_i} \bmod p\}$. Also, PK protocols can be used for proving the knowledge and equivalence of discrete logarithms of different representations. The protocol $PK\{(w_1, w_3) : c_1 = g_1^{w_1} g_2^{w_2} g_3^{w_3} \wedge c_2 = g_1^{w_2} g_3^{w_3} \bmod p\}$ is a simple example in CS notation. The number of possible variations is unlimited. The work [8] can be taken as a fundamental reference for the construction of PK protocols.

By using the Fiat-Shamir heuristic [27], all the PK protocols can run non-interactively. Then, a hash $\mathcal{H}$ of all protocol parameters is used. All protocols shown in the Appendix are non-interactive. This adaptation leads to signature schemes. By taking our simple $PKDL$ protocol from Sect. 2, we can get a digital signature protocol where w works as a private key. The protocol uses a hash

function $\mathcal{H}$ of the message and all protocol parameters. For reference, the signature proof of knowledge protocol (SPK) is depicted in Fig. 17 in Appendix. All PK protocols can be easily adapted to signatures using this approach [27].

In previous sections, we considered only examples based on the simple DSA group [23]. But also different groups can be used in PK protocols (e.g., RSA group [22], Okamoto-Uchiyama (OU) group [28], etc.). Still, the atomic operations of these protocols remain the same. Namely, modular arithmetic, random number generation and hash functions are used. That is the reason why we implemented these atomic operations in our benchmarks. Based on their performance, we can compute the actual performance of PK protocols and subsequently the performance of advanced systems based on PK protocols.

3 Selected Devices and Benchmark Settings

This chapter contains the information about the benchmark settings and about the software/hardware we used.

3.1 Selected Devices

The evaluation of cryptographic primitives was carried out using all major smartcard platforms, namely JavaCards [3], .NET cards [4] and MultOS cards [5]. Furthermore, we implemented the benchmark tests on the Android platform, namely on Android smart-phones and an Android tablet.

JavaCards. JavaCard platform [3] provides a development and runtime environment for applications (called applets) written in Java. In our benchmarks, we used Oberthur Technologies ID-One Cosmo V7.0-A [29,30] and Gemalto TOP

Table 1. The specification of the JavaCards used in our benchmarks.

	Software specifications	
Card type	Oberthur ID-One V7.0-A	Gemalto TOP IM GX4
Type	JavaCard	JavaCard
Transfer protocol	T=0, T=1	T=0, T=1
Asymmetric crypto	RSA upto 2048, EC upto 521 b	RSA upto 2048 bits
Symmetric crypto	DES, 3DES, AES	3DES, AES
Hash	SHA1, SHA2	SHA1
	Hardware specifications	
Chip	Atmel AT90SC256144RCFT	S3CC9TC
CPU	8/16 bit	16 bit
Internal/External clock	40 MHz/3.4 MHz	Unknown
RAM memory	8 kB	10 kB
ROM/EEPROM	256 kB/144 kB	384 kB/74 kB
Temperature range	−25 °C to +85 °C	−25 °C to +85 °C
Modular arithmetic API	No	No

Table 2. The specification of the .NET cards and the MultOS cards used in benchmarks.

Software specifications			
OS type	.NET	MultOS	MultOS
Card type	.NET V2+	ML2-80K-65	ML3-36K-R1
Asymmetric crypto	RSA 2048 bits	RSA 2048, EC 384 bits	RSA 2048, EC 512 bits
Symmetric crypto	3DES, AES	DES, 3DES, AES	DES, 3DES, AES
Hash	SHA1, SHA2, MD5	SHA1, SHA2	SHA1, SHA2
Hardware specifications			
Chip	SLE 88CFX4000P	SLE66CLX800PEM	SLE78CLXxxxPM
CPU	32 bit	16 bit	16 bit
Internal/External clock	66 MHz/10 MHz	30 MHz/7.5 MHz	33 MHz/7.5 MHz
RAM memory	16 kB	702 + 9604 B	1088 + 960 B
ROM/EEPROM	80 kB/400 kB	236 kB/78 kB	280 kB/60 kB
Temperature range	−25 °C to +85 °C	−25 °C to +85 °C	−25 °C to +85 °C
Modular API	No	Yes	Yes

IM GX4 [31] cards. The hardware specification of these cards is described in Table 1.

.NET Smart-Cards. .NET smart-card platform [4] provides very similar features as JavaCards for applications developed using any language of the .NET framework. In our benchmarks, we used the Gemalto .NET V2+ cards. The hardware specification of these cards is described in Table 2.

MultOS Smart-cards. The last smart-card platform we used for benchmarking is the MultOS platform [5]. In comparison to JavaCard and .NET cards, MultOS allows the development of applications in both high level languages (Java and C) and assembly language. This provides developers with much wider opportunities and better access to hardware. In particular, only the MultOS cards allow the direct big-integer modular operations through the default API. The hardware specification of MultOS ML2-80K-65 and ML3-36K-R1 cards is described in Table 2.

Mobile Devices. The Android devices form a different group which is incomparable to smart-cards. While smart-cards are very resource-limited devices with extremely low RAM and slow CPUs, the mobile phones and tablets resemble more classical PCs. They have strong CPUs with frequency over 1 GHz and enough RAM (hundreds of megabytes). Still, these devices are extremely mobile and very popular for personal electronic transactions. Due to this reason, we included them to our benchmarks. The hardware of selected Android devices is described in Table 3.

Table 3. The specification of the Android devices used in our benchmarks.

	Software specifications		
Device type	Samsung Galaxy S i9000	Samsung Galaxy Nexus I9250M	ASUS TF 300T
Android version	v2.1 (Eclair)	v4.0 (ICS)	v4.0 (ICS)
	Hardware specifications		
Chip	Cortex-A8	Dual-core Cortex-A9	Quad-core Cortex-A9
Frequency	1 GHz/45 nm	1.2 GHz/45 nm	1.2 GHz/45 nm
GPU	PowerVR SGX540	PowerVR SGX540	ULP GeForce
RAM memory	512 MB	1024 MB	1024 MB
ROM/Storage	2 GB/8(16) GB	2 / 16 GB	2 GB/16(32) GB

3.2 Measured Operations and Keylengths

In the Sect. 2, we showed the cryptographic commitments and proof of knowledge protocols. We included only simple examples to illustrate these primitives. Nevertheless, these basic constructions can be modularly compiled into advanced systems. The discrete logarithm commitments, proof of knowledge of discrete logarithm protocols and proofs of discrete logarithm equivalence protocols are the building blocks of many complex modern systems [17–19]. But still, even the complex systems are based on the same atomic operations as the primitives selected by us. It can be observed from Sect. 2 that only random number generation, hash functions and big-integer modular arithmetic operations are needed for all selected protocols. Namely, the following operations are required.

- **RNG - Random Number Generation**: on all platforms and devices, we measured the time of generation of large random numbers of length 160 bits (**RNG_160** operation) and 560 bits (**RNG_560** operation).

- **Hash Functions**: on all platforms and devices, we measured the time of computation of following hash functions.
  - **SHA1_4256**: SHA1 of 4256 bit random data[2]
  - **SHA1_7328**: SHA1 of 7328 bit random data
  - **SHA1_20000**: SHA1 of 20000 bit random data
  - **SHA2_8448**: SHA2 of 8448 bit random data
  - **SHA2_14592**: SHA2 of 14592 bit random data
  - **SHA2_20000**: SHA2 of 20000 bit random data

- **Big-Integer Modular Arithmetic Operations**: it can be observed from our cryptographic overview in Sect. 2 that the proof of knowledge protocols heavily rely on arithmetic operations in groups where the discrete logarithm operation is hard to compute. Namely, modular operations with moduli in orders of thousand bits are required. These operations are usually available

[2] The size of data hashed reflects the requirements of PK protocols.

on the PC platform in the form of BigInt libraries (such as OpenSSL, Bouncy Castle, etc.). Unfortunately, these libraries are missing on smart-cards. Only the MultOS platform supports direct modular operations. Thus, the following operations were implemented and measured on all selected platforms and devices. The bit-lengths of moduli and operands were selected according to the most popular group sizes in cryptography (1024 and 2048 bit modulus).

- **MExp1024_160:** Modular Exponentiation with 1024 b modulus and 160 b exponent
- **MExp1024_368:** Modular Exponentiation with 1024 b modulus and 368 b exponent
- **MExp2048_160:** Modular Exponentiation with 2048 b modulus and 160 b exponent
- **MExp2048_560:** Modular Exponentiation with 2048 b modulus and 560 b exponent
- **MMult1024:** Modular Multiplication with 1024 b modulus and operands
- **MMult2048:** Modular Multiplication with 2048 b modulus and operands

– **Big-Integer Arithmetic Operations**: additionally to modular operations, some non-modular (plain) big-integer operations were implemented as they are contained in PK protocols which operate in hidden order groups.
  - **Mult320**: Multiplication of two 320 b numbers
  - **Sub400**: Subtraction of two 400 b numbers

Although the above selected bit-length combinations do not include all the variants used in today's cryptographic schemes, they represent a sample which can be further interpolated to get the estimation of other bit-lengths. Thus, an estimate of smart-card performance of any new protocol, which is based on above operations, can be created.

3.3 Benchmark Environment

The hardware selected for our benchmarks is described in Sect. 3. The operations measured on the hardware are listed in the previous Sect. 3.2. We measured the time necessary for the computation of each operation 25 times. We present the arithmetic mean of these values. The resulting time does not include the time of communication with the device (sending inputs and receiving results). The code was implemented by a single person on smart-cards and by a single person on Android devices. Thus, the influence of different programming styles is eliminated. We tried to use the default API of our cards as much as possible. To increase the speed of computation, we used the RSA encryption method to implement modular exponentiation. For many operations (e.g., for modular arithmetic), only some cards, namely those running MultOS, were providing the necessary interface. On the rest, we had to implement our methods.

4 Benchmark Results

We divided our results into a smart-card section and an Android section. The graphs in the next two sections show the time in milliseconds of the operations specified in the Sect. 3.2 above.

4.1 Benchmarks on Smart-card Devices

Figures 1, 2, 3, 4, 5, 6 and 7 show the time in milliseconds of operations specified in captions.

4.2 Benchmarks on Android Mobile Devices

Figures 8, 9, 10, 11, 12, 13 and 14 show the time in milliseconds of operations specified in captions.

4.3 Results Analysis

Smart-Cards. It was possible to implement all required operations on all selected cards with the exception of MultOS ML2-80K-65 card which is lacking the support of SHA2 and 2048 b modular exponentiation. In many operations, the JavaC-

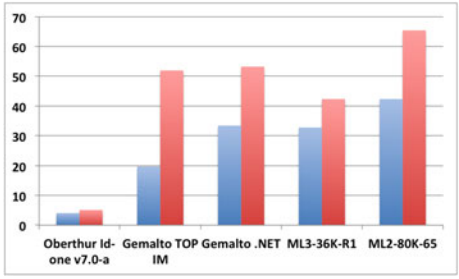

Fig. 1. **RNG_160** (blue) and **RNG_560** (red)

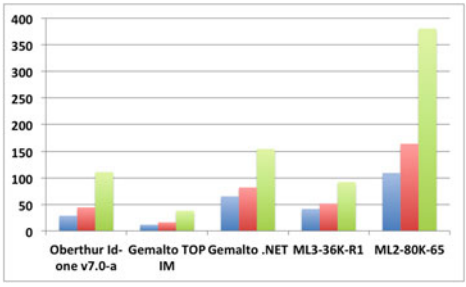

Fig. 2. **SHA1_4256** (blue), **SHA1_7328** (red) and **SHA1_20000** (green)

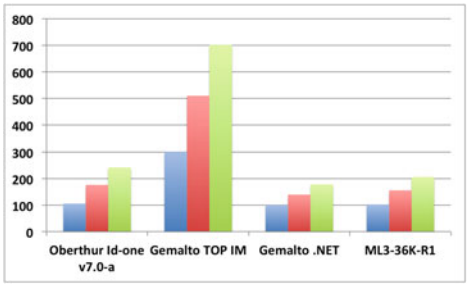

Fig. 3. **SHA2_8448** (blue), **SHA2_14592** (red) and **SHA2_20000** (green)

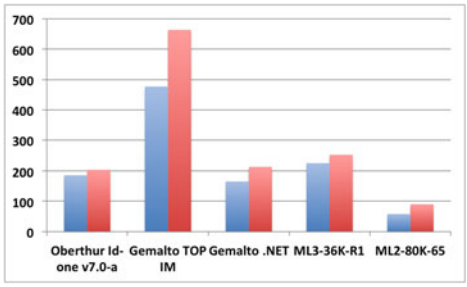

Fig. 4. **MExp1024_160** (blue) and **MExp1024_368** (red)

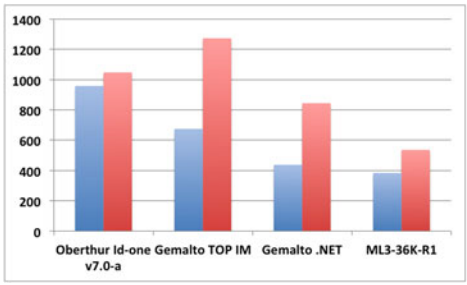

Fig. 5. **MExp2048_160** (blue) and **MExp2048_560** (red)

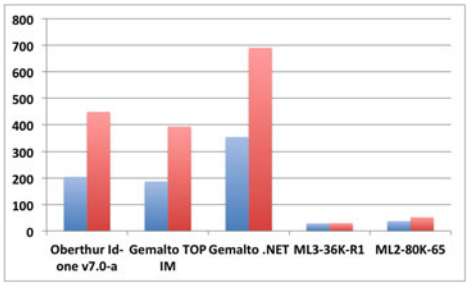

Fig. 6. **MMult1024** (blue), **MMult2048** (red)

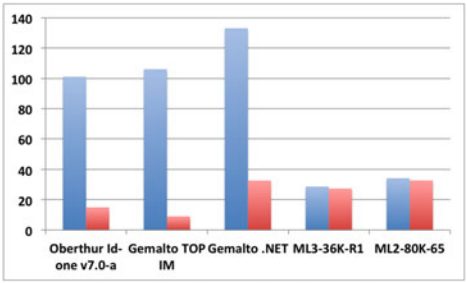

Fig. 7. **Mult320** (blue) and **Sub400** (red)

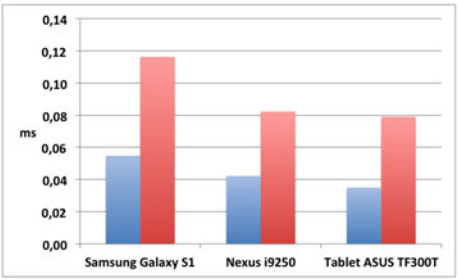

Fig. 8. **RNG_160** (blue) and **RNG_560** (red)

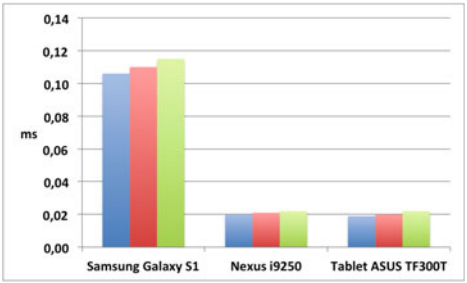

Fig. 9. **SHA1_4256** (blue), **SHA1_7328** (red) and **SHA1_20000** (green)

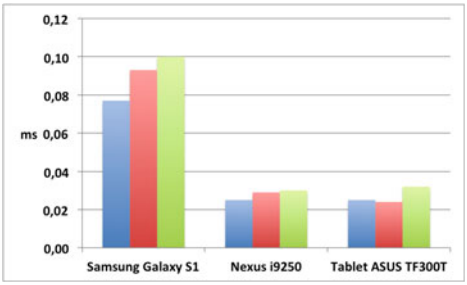

Fig. 10. **SHA2_8448** (blue), **SHA2_14592** (red) and **SHA2_20000** (green)

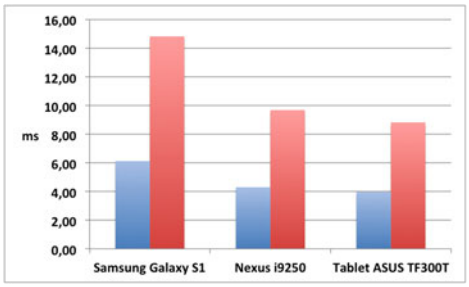

Fig. 11. **MExp1024_160** (blue) and **MExp1024_368** (red)

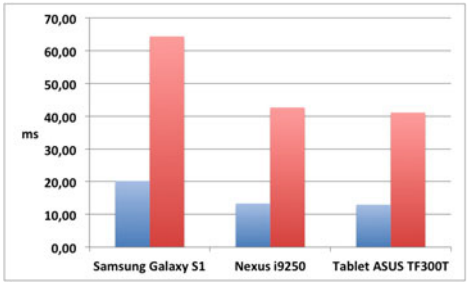

Fig. 12. **MExp2048_160** (blue) and **MExp2048_560** (red)

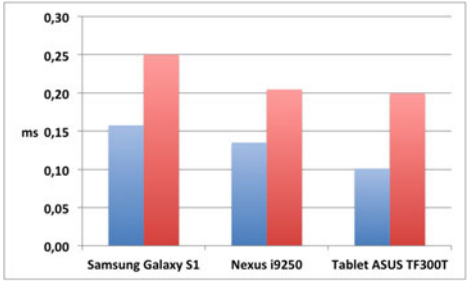

Fig. 13. **MMult1024** (blue), **MMult2048** (red)

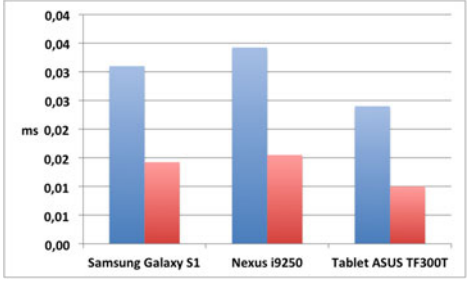

Fig. 14. **Mult320** (blue) and **Sub400** (red)

ard Oberthur ID-one v7.0a is very fast (in particular, in random number generation and 1024 b modular exponentiation). Often, the bit-length of inputs (cryptographic group size) does play a significant role, for example in the case of modular exponentiation. Thus, we recommend to plan ahead before implementing and choose the right balance between speed and security (group size). Even with modern smart-cards, operations in 2048 b groups might be too demanding. When implementing complex privacy-enhancing schemes, operations in 2048 b groups would be probably too slow. Also, a big difference among cards appears when the modular multiplication and non-modular operations are needed. This is the case of all PK protocols where a group with unknown order is used (such as RSA group [22], OU group [28]). Then, the MultOS cards are much faster than the rest due to their direct support of these operations in API, in particular due to their built-in support of accelerated modular multiplication.

Android Devices. It is no surprise that most operations are several hundred times faster on Android devices than on smart-cards. All primitives can be easily implemented on Android. Due to the high performance, we recommend using larger (and safer) 2048-bit groups and more recent primitives (e.g., SHA-2 instead of SHA-1 or MD5).

4.4 Performance Estimation of Selected Protocols and Schemes

Using the results of our benchmarks, we estimated the theoretical performance of the protocols introduced in the Sect. 2 and of some well-known privacy-enhancing schemes like Idemix of IBM [13], U-Prove of Microsoft [18] and HM12 [19]. All protocols are evaluated using 1024 bit groups and 160 bit secrets. All the scheme estimates include only the time of operations needed for proving the ownership of an anonymous token (we use the same approach as in [13]) and do not include any communication/management overhead. Furthermore, we used the closest bit-length of inputs. Thus, the numbers in Table 4 should be considered estimates only.

We created the estimates using our implementation of atomic operations. From the knowledge of the construction of the advanced protocols and the knowledge of performance of underlying operations, we were able to predict the performance of protocols. To find out the correctness of our estimates, we compared our results with existing, real implementations. Since they use smart-cards of different specifications, the comparison is rough only. The IBM's Idemix has been previously implemented on JavaCards [13]. The proving protocol of the 1280 bit version took 7.5 s. Our estimates of the 1024 b version are 4.5 and 9.4 s, depending on the concrete type of our JavaCard. The Microsoft's U-Prove scheme has been implemented on the MultOS platform [5]. The proving protocol took 0.55 son an unspecified MultOS-family card. Our estimates on our MultOS cards are 0.63 and 0.82 s, depending on the concrete type of the MultOS card. Based on these results, we consider our estimates highly accurate. Using our benchmarks, it is possible to easily predict the approximate time of newly designed protocols or cryptographic schemes.

Table 4. Performance estimation based on benchmarks.

	Time in ms							
	S1	**S2**	**S3**	**S4**	**S5**	**A1**	**A2**	**A3**
$c = g^w$ (DL commitment)	186	476	165	226	58	6	4	4
$c = g^w h^r$ (Pedersen commitment)	580	1161	717	513	195	12	9	8
$PK\{w : c = g^w\}$	325	830	433	352	222	15	10	9
$PK\{w : c_1 = g_1^w \wedge c_2 = g_2^w\}$	529	1494	646	605	313	30	20	18
$SPK\{w : c = g^w\}(m)$	354	842	498	393	332	15	10	9
Idemix	4519	9433	7270	4219	4208	153	100	91
U-Prove	837	1618	1295	827	633	13	9	8
HM12	2540	6016	3312	2509	1467	102	68	62

Glossary:
S1: Oberthur Technologies ID-One Cosmo V7.0-A
S2: Gemalto TOP IM GX4
S3: Gemalto .NET V2+
S4: MultOS ML2-80K-65
S5: MultOS ML3-36K-R1
A1: Samsung Galaxy S i9000 (smart-phone)
A2: Samsung Galaxy Nexus I9250M (smart-phone)
A3: ASUS TF 300T (tablet)

5 Conclusion

In this paper, we provide the performance evaluation of modern cryptographic primitives on smart-cards and mobile devices. In particular, selected atomic operations which are the core of many privacy-enhancing protocols and schemes are implemented on all major programmable smart-card platforms and on the Android platform. The results can be used for the evaluation of many existing and newly appearing schemes. Using the results of implementation of all operations used in PK protocols, it is possible to predict the performance of any protocol or scheme which is composed of DL-based commitments and/or DL-based proof of knowledge protocols. In particular, it is possible to predict the performance of very popular computational zero-knowledge protocols.

Even with the fastest smart-cards on the market, it is quite difficult to achieve reasonable execution times. Though, with the right choice of hardware, in particular, with hardware-accelerated cards, it is possible.

We showed our performance estimates of today's most preferred privacy-enhancing anonymous credential schemes on all 8 devices. When compared to existing implementations, we almost match the real performance when similar hardware is used. Thus, our benchmarks can be used by cryptography designers to easily predict the performance of their protocols and schemes before implementing on smart-cards and Android devices.

Acknowledgment. This research work is funded by projects SIX CZ.1.05/2.1.00/03.007; the Technology Agency of the Czech Republic projects TA02011260 and TA03010818; the Ministry of Industry and Trade of the Czech Republic project FR-TI4/647.

Appendix

Simple examples of Proof of Knowledge (PK) protocols. All operations are in a group $\mathbb{Z}_p^*$ of order q where discrete logarithm is hard to compute and l_1, l_2 are security parameters. More information about PK protocols in [8].

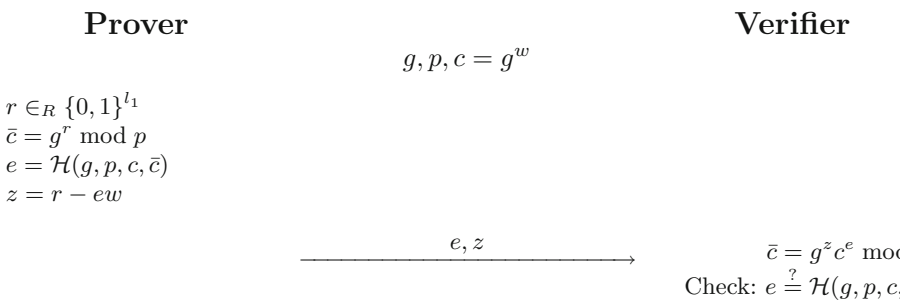

Fig. 15. Schnorr's proof of knowledge of discrete logarithm protocol $PK\{w : c = g^w\}$.

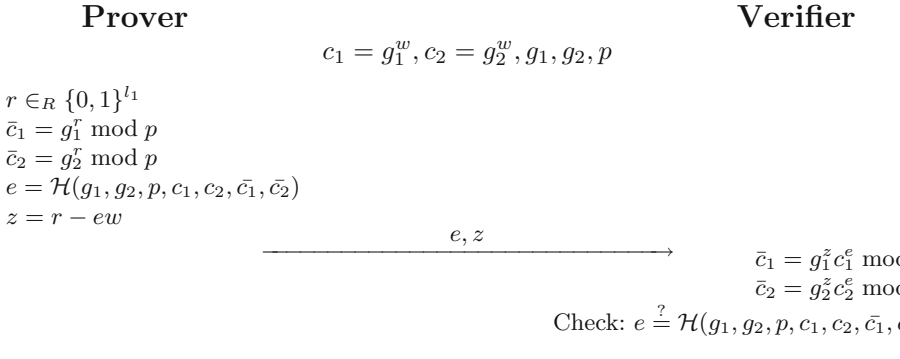

Fig. 16. Proof of discrete logarithm equivalence $PK\{w : c_1 = g_1^w \wedge c_2 = g_2^w\}$.

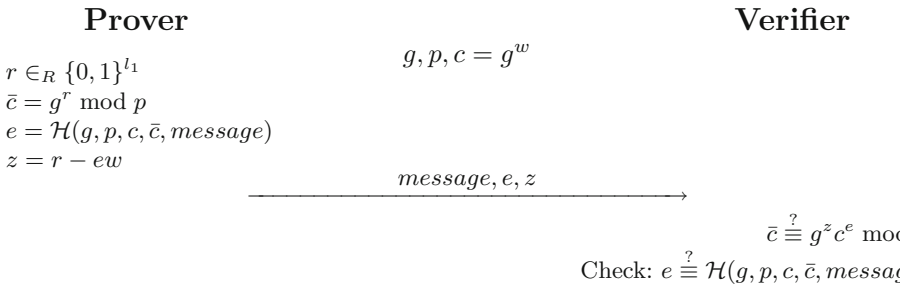

Fig. 17. Schnorr's signature $SPK\{w : c = g^w\}(message)$.

References

1. Eisenbarth, T., et al.: Compact implementation and performance evaluation of block ciphers in attiny devices. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 172–187. Springer, Heidelberg (2012)
2. Balasch, J., Ege, B., Eisenbarth, T., Gérard, B., Gong, Z., Güneysu, T., Heyse, S., Kerckhof, S., Koeune, F., Plos, T., Pöppelmann, T., Regazzoni, F., Standaert, F.X., Assche, G.V., Keer, R.V., van Oldeneel tot Oldenzeel, L., von Maurich, I.: Compact implementation and performance evaluation of hash functions in attiny devices. IACR Cryptology ePrint Archive (2012)
3. Oracle: Javacard. http://www.oracle.com/technetwork/java/javacard/downloads/index.html (2013)
4. Gemalto: .net card. http://www.gemalto.com/products/dotnet_card/ (2013)
5. MultOS: Multos card. http://www.multos.com (2013)
6. Deloitte: The deloitte open mobile survey 2012. http://www.deloitte.com/assets/Dcom-Norway/Local%20Assets/Documents/Publikasjoner%202012/deloitte_openmobile2012.pdf (2012)
7. Cramer, R.: Modular design of secure, yet practical cryptographic protocols. Ph.D. thesis, University of Amsterdam (1996)
8. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Technical report (1997)
9. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
10. Chaum, D., Van Heyst, E.: Group signatures. In: Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques, EUROCRYPT'91, pp. 257–265. Springer, Heidelberg (1991)
11. Stadler, M.A., Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 32–46. Springer, Heidelberg (1998)
12. Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 93. Springer, Heidelberg (2001)
13. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard java card. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, pp. 600–610. ACM, New York (2009)
14. Mostowski, W., Vullers, P.: Efficient u-prove implementation for anonymous credentials on smart cards. In: Rajarajan, M., Piper, F., Wang, H., Kesidis, G. (eds.) SecureComm 2011. LNICST, vol. 96, pp. 243–260. Springer, Heidelberg (2012)
15. Hajny, J.: Anonymous authentication for smartcards. Radioengineering **19**(2), 363–368 (2010)
16. Malina, L., Hajny, J.: Accelerated modular arithmetic for low-performance devices. In: 34th International Conference on Telecommunications and Signal Processing, pp. 131–135. IEEE (2011)
17. Camenisch, J., et al.: Specification of the identity mixer cryptographic library. Technical report. http://domino.research.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/eeb54ff3b91c1d648525759b004fbbb1?OpenDocument (2010)
18. Paquin, C.: U-prove cryptographic specification v1.1. Technical report. http://research.microsoft.com/apps/pubs/default.aspx?id=166969 (2011)

19. Hajny, J., Malina, L.: Unlinkable attribute-based credentials with practical revocation on smart-cards. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 62–76. Springer, Heidelberg (2013)
20. FIPS: Data encryption standard. In: Federal Information Processing Standards Publication, FIPS PUB 46, 46–2 (1977)
21. FIPS: Advanced encryption standard (aes). In: Federal Information Processing Standards Publication, FIPS PUB 197, pp. 1–47 (2001)
22. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**, 120–126 (1978)
23. National Institute of Standards and Technology (U.S.) : Digital Signature Standard (DSS) [electronic resource]. U.S. Deptartment of Commerce, National Institute of Standards and Technology, Gaithersburg (2009)
24. Rivest, R.: The md5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt (1992)
25. FIPS: Secure hash standard (shs) (2012)
26. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**, 161–174 (1991)
27. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
28. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 308–318. Springer, Heidelberg (1998)
29. Id-one cosmo v7.0: Technical report, French Network and Information Security Agency (Agence Nationale de la Scurit des Systmes dInformation (ANSSI)). http://www.ssi.gouv.fr/IMG/certificat/anssi-cc-cible_2009-36en.pdf (2009)
30. Atmel: At90sc256144rcft datasheet. http://datasheet.elcodis.com/pdf2/104/7/1040758/at90sc256144rcft.pdf (2007)
31. NIST: Gemxpresso r4 e36/e72 pk—multiapp id 36k/72k—top im gx4. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp771.pdf (2009)