Lightweight Password Hashing Scheme for Embedded Systems

George Hatzivasilis[1], Ioannis Papaefstathiou[1], Charalampos Manifavas[2], and Ioannis Askoxylakis[3]

[1] Dept. of Electronic & Computer Engineering, Technical University of Crete,
Chania, Crete, Greece
gchatzivasilis@isc.tuc.gr, ygp@mhl.tuc.gr

[2] Dept. of Informatics Engineering, Technological Educational Institute of Crete,
Heraklion, Crete, Greece
harryman@ie.teicrete.gr

[3] Foundation for Research and Technology – Hellas (FORTH), Heraklion, Crete,
Greece
asko@ics.forth.gr

Abstract. Passwords constitute the main mean for authentication in computer systems. In order to maintain the user-related information at the service provider end, password hashing schemes (PHS) are utilized. The limited and old-fashioned solutions led the international cryptographic community to conduct the Password Hashing Competition (PHC). The competition will propose a small portfolio of schemes suitable for widespread usage until 2015. Embedded systems form a special application domain, utilizing devices with inherent computational limitations. Lightweight cryptography focuses in designing schemes for such devices and targets moderate levels of security. In this paper, a lightweight poly PHS suitable for lightweight cryptography is presented. At first, we design two lightweight versions of the PHC schemes Catena and PolyPassHash. Then, we integrate them and implement the proposed scheme – called LightPolyPHS. The schemes are applied on a MANET with BeagleBone embedded devices and a fair comparison with similar proposals on mainstream computer is presented.

Keywords: password hashing · PHC · Catena · PolyPassHash · lightweight cryptography · LWC · embedded systems · BeagleBone

1 Introduction

Attacks on widely known organizations, like SONY [1] and LinkedIn [2], have exposed mounts of user accounts and credentials. The poor password protection practises [3] are exploited by the attackers in order to recover the user passwords from the stolen data. These attacks lead to negative and significant loss in the vendor's market value [4].

Advanced password hashing schemes (PHS) are proposed to fortify the secure maintenance of such information. PBKDF2 [5], bcrypt [6] and scrypt [7] are currently the most common solutions.

However, the evolving parallel computing and hardware dedicated devices enable attacks [8] that overcome the PHS protection. GPUs, FPGAs and ASICs implement efficient password crackers that try out several attempts in parallel, gaining a significant boost in disclosing the user information. PBKDF2 and bcrypt are vulnerable to such attacks.

Memory-hard PHSs can counter password scrambling. The memory elements in parallel platforms are considered expensive. All parallel components share the same memory and the access to it is bounded. Thus, attackers are significantly slowed-down when PHSs with high memory requirements are applied. The goal is to derive password scrambling on parallel cores not much faster than it is on single cores. scrypt is the current solution of memory-hard PHS. Unfortunately, it remains vulnerable to other attacks, like cache-timing [9] and garbage-collector attacks [10].

The limited and old-fashioned solutions led the international cryptographic community to conduct the Password Hashing Competition (PHC) [11] in 2013. It targets in modern and secure designs for password hashing, with 22 initial candidates being submitted. In 2014, 9 finalists were selected based on security, efficiency, simplicity, and the extra features that they provide. A comprehensive survey and benchmark analysis of the 22 PHC submissions and the 3 current solutions for password hashing is presented in [12]. In 2015, a small portfolio of schemes will be announced based on further performance and security analysis. The winners are expected to become ”de facto” standards and be further examined by organizations like NIST [13] for formal standardization.

PHSs are applied in several domains (e.g. general applications on mainstream computers, web applications and embedded systems) with diverse features and properties. The candidate scheme must comply with them. Typically, a PHS utilizes core cryptographic primitives, such as block ciphers and hash functions, that constitute the main computational components of the scheme.

The mainstream cryptographic solutions provide high levels of security, ignoring the requirements of resource constrained devices. The research field of lightweight cryptography (LWC) focuses in designing schemes for devices with constrained capabilities in processing, power supply, connectivity, hardware and software. They are mainly applied in embedded systems that are deployed in pervasive and ubiquitous computing [14]. Security is just a part of the whole functionality and the lightweight designs consume low computational resources and memory [15]. In case of most constraint devices (e.g. sensors) only a few KBs of memory are devoted to provide moderate level of security [16].

Regarding passwords, embedded systems maintain a small amount of authentication-related data. Device-to-device and short-term communication forms the most common interaction (e.g. in wireless sensor networks) [17], making session key deviation from passwords a desirable goal to enhance security. The garbage-collector attacks [10] can be countered by build-in memory safety techniques, specifically designed for embedded applications [18].

In this paper, we present LightPolyPHS – a lightweight poly PHS for embedded devices and LWC. To our knowledge this is the first scheme that targets constrained devices. Section 2, introduces the background theory and related work regarding passwords and LWC. Section 3, analyses the LightPolyPHS design and its subcomponents. In section 4, the proposed scheme is applied and evaluated on real embedded devices. A comparative analysis is held with similar schemes on mainstream computers. Finally, section 5 concludes.

2 Background Theory and Related Work

2.1 Password Hashing

User passwords are human-memorable secrets [19], consisting of 8-12 printable characters and form the main mean for authentication in computer systems. The service provider maintains a pair of the user's name and password-related information for each active account. To authenticate himself and login the service, the user must inputs this information first (e.g. [20, 21]).

Passwords can also be used for the generation of cryptographic keys. Key Deviation Functions (KDF) [22] parse a password and derive one or more user-related keys. These keys are used on cryptographic operations, like session communication encryption [23, 24].

Ordinary passwords are 8 characters long (8 bytes). The deriving secrets may produce low entropy and be vulnerable to relative attacks. In exhaustive search, an attacker tries out all character combinations until he finds the right password for an examined username. Then, he owns the relevant account, like the legitimate user does. Even newer user-drawn graphical passwords suffer from low-entropy properties and similar security issues, as they provide an average security of 4-5 bytes [25].

The typical method to counter these attacks with PHSs, is key stretching. A hash function parses the password and produces a fixed-length output, acting as the new password. The hash password is longer (usually 32-64 bytes long), making the attacks less feasible. The hash function is iterating several times to further fortify the hash result. The attacker is slowed down by a factor of 2^{i+o}, where i is the iteration count and o is the number of the output bits. However, the user is also slowed down. Thus, the key stretching parameters are also bounded by the user's tolerance to compute a robust hash password.

In modern services a high volume of users must be verified simultaneously. The load on the server may become unmanageable and lead to denial of service. Server relief (SR) protocols are established and balance the total effort between clients and the server. A client may perform part of the PHS computations (e.g. the first PHS iterations) while the server performs the rest steps and the account verification.

The server might need to increase security (e.g. increase the hash size or PHS iterations). Hash password upgrade techniques independent from the user

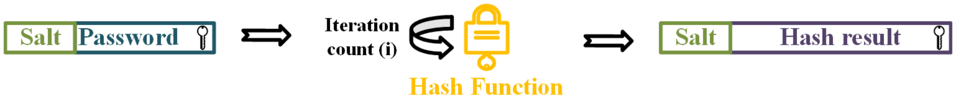


Fig. 1. PHS generic scheme.

(HUIU) enhances the convenience of the user and enables the seamless operation of the service. The server upgrades the security of the stored hash passwords without prior-knowledge of the initial passwords or the user's involvement.

It is quite common for a user to utilize the same password in different services or many users of a service to have the same password. The hash passwords would be the same too. The disclosure of a single account erases security issues for the rest ones. To prevent this correlation a small parameter of random bytes, called salt (usually 8 bytes long), is utilized. The salt also hardens dictionary (try hundreds of likely possibilities to determine the secret) [9] and rainbow table attacks (ability to use tables of precomputed hashes) [10]. Typically, the salt is generated when the user account is created and is concatenated with the password during hashing. Thus, the same password produces different hash passwords. At the server-end, the salt is stored in plaintext along with the hashed password. They are used in the authentication process to validate the password of a login request. Figure 1, illustrates the generic PHS.

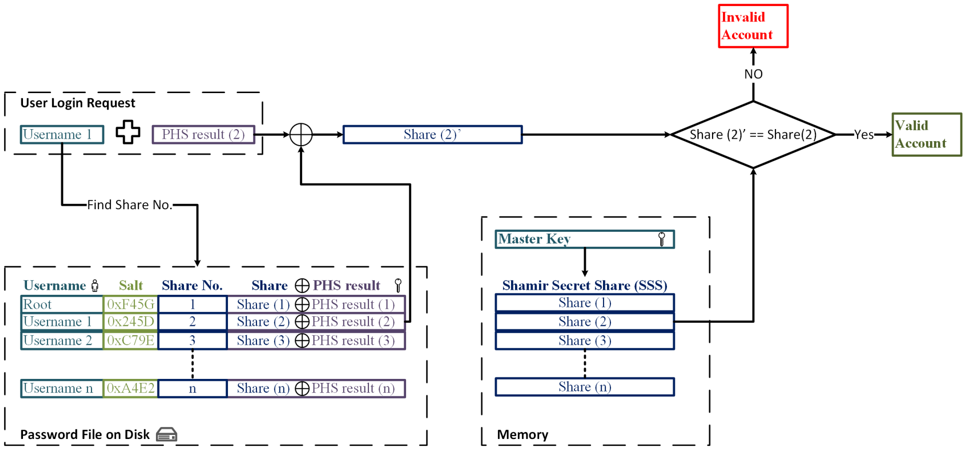
2.2 Poly Password Hashing

Strong PHS can protect the password data that are maintained at the service-end. However, attackers have proven themselves adept at cracking large amounts of passwords once the stored data is compromised.

To further fortify security and harden attackers' cracking capabilities, poly (many) password hashing (PPH) schemes have been recently proposed. They leverage cryptographic hashing and threshold cryptography by combining strong PHS with shares.

Cryptographic hashing and PHSs are described in the previous subsection. A cryptographic (k, n)-threshold scheme protects secret information, by deriving n different shares from this information. The threshold determines how any k shares out of total n can recover the secret information. If fewer than k shares are known, no secret information is disclosed.

The Shamir Secret Sharing (SSS) [26] is a fundamental threshold scheme in this domain. It computes $k-1$ random coefficients for a $k-1$ degree polynomial $f(x)$ in a finite field (e.g. GF-256 or GF-65536). The kth term comprises the secret (usually the constant term of the polynomial). The share is identified by a share value x, taking values between 1 and the order of the field. The share x is the polynomial value of $f(x)$. The secret can be reconstructed by interpolating the values of k shares to find the constant term of the polynomial (i.e., the secret). Interpolation is computationally optimized and only the constant term is revealed.



Username	Salt	Share No.	Share ⊕ PHS result
Root	0xF45G	1	Share (1) ⊕ PHS result (1)
Username 1	0x245D	2	Share (2) ⊕ PHS result (2)
Username 2	0xC79E	3	Share (3) ⊕ PHS result (3)
		⋮	
Username n	0xA4E2	n	Share (n) ⊕ PHS result (n)

Fig. 2. Poly password hashing scheme. The figure illustrates the elements that are stored in memory and hard disk and the account verification for a login request.

In the PPH domain, there is one share for each account. The share is XORed with the relevant PHS result and is maintained by the server (instead of the pure PHS result). The shares are derived from a master key. This key is only known to the service provider and is not stored on disk in order to prevent attacks that would disclose the key along with the stolen password data. When the server starts, k clients must login and be correctly verified in order to reconstruct the shares. Implementations of SSS provide integrity check mechanisms to detect if incorrect shares are parsed. After this startup phase, the server operates in the ordinary manner. Figure 2, illustrates the generic PPH scheme and the account verification of a login request after initialization.

The attacker has to crack a threshold of password hashes before being able to recover passwords. At a small additional cost by the server, security increases by many orders of magnitude.

Poly password hashing is easily implemented and deployed on a server without any changes to clients and can be integrated to current forms of authentication (e.g. two factor authentication, hardware tokens and fingerprint authentication). It is also efficient in terms of storage, memory and computational demands.

2.3 Password Hashing Competition

Secure cryptographic hash functions or HMACs constitute the most common solution for PHSs and KDFs. PBKDF2, bcrypt and scrypt are currently the widely-used PHSs and KDFs for mainstream applications.

The Password-Based Key Derivation Function 2 (PBKDF2) [5] is the only standardized scheme (RSA Laboratories' Public-Key Cryptography Standards (PKCS) series (PKCS #5 v2.0) [27] and the RFC 2898 [5]). The input password and salt are processed by an HMAC. The standard, which was established 15 years ago, recommended a minimum of 1000 iterations but it is not adequate for current applications. PBKDF2 is not memory-hard and can be implemented as a small circuit wit low RAM requirements. This is evinced in a main drawback as cheap brute-force attacks are enabled on GPUs and ASICs.

bcrypt [6] is based on the block cipher Blowfish [28] and is the default PHS of the BSD operating system. The iteration count is a power of 2 and is periodically increased to counter the increasing computation power of the attackers. It uses 4KB RAM and is slightly stronger than PBKDF2 in defending attacks on parallel computing platforms and dedicated hardware devices. However, these memory requirements render efficient attacks on FPGAs.

scrypt [7] was announced as an Internet Draft by the IETF in 2012, with the intention to become an informational RFC [29]. It utilizes the PBKDF2 and the stream cipher Salsa [30] and uses arbitrarily large amounts of memory. scrypt is the most resistant widely-used scheme. The cost of a hardware brute force attack is considered around 4000 and 20000 times larger than in bcrypt and PBKDF2 respectively. However, the huge memory requirements can derive denial-of-service (DoS) attacks on servers, when large amounts of simultaneous login requests are handled. Also, scrypt is vulnerable to new types of attacks, like cache-timing [9] and garbage-collector attacks [10].

Password Hashing Competition (PHC) [11] advances our knowledge in designing secure and efficient PHSs and KDFs. At the first round 22 new PHSs [31] were evaluated in terms of security, performance and flexibility. 9 state-of-the-art finalists are announced and a small portfolio of them will be selected as the winners in 2015. A survey and benchmark analysis of the 3 aforementioned widely-used PHSs and the 22 candidates is presented in [12].

All submissions implement a common API. The parameters *t_cost* and *m_cost* are introduced to adjust the timing and memory requirements of each scheme respectively. The defender adjusts the PHS iteration count and memory requirements to design secure schemes.

The finalist Catena is one of the most notable submissions and is intended to be included in the winners list. It implements the full functionality of PHS, KDF, SR, and HUIU, is well-documented and analysed, and is one of the most efficient candidate in terms of execution time and memory usage. Catena exhibits low code size and memory requirements, making it suitable for embedded systems.

A PPH scheme, called PolyPassHash, is also presented in the competition. It is actually a protocol that recovers a symmetric key used to encrypt passwords and does not constitute a pure PHS. Thus, it is not included in the finalists. Still, PolyPassHash demonstrates state-of-the-art features regarding PPH and is efficient in terms of storage, memory and computational requirements.

We utilize the PHC candidates Catena and PolyPassHash to design our lightweight proposal. Both schemes are analysed in the following section.

2.4 Lightweight Cryptographic Mechanisms

Traditional cryptography targets high level of security. The main primitive types that are investigated in this paper include block ciphers and hash functions.

The block cipher AES[32] is considered a landmark in this field. Camellia [33], IDEA [34], KASUMI [34] and Blowfish are other widely used ciphers. In PHC, AES and Blowfish are mainly utilized for cryptographic operations.

There are many standardized or widely-used hash functions for mainstream applications, like MD5 [35], SHA1 [36], SHA2[37] and Whirlpool [38]. In 2012, the NIST's SHA3 competition [39] establish a new function to replace the older SHA1 and SHA2. The SHA3 competition helped our understanding of hash functions and derive a new design trend of hash functions with sponge constructions. The winner function Keccak and the finalist BLAKE[40] adopt this strategy. SHA1, SHA2, SHA3 and BLAKE are widely used by PHC candidates.

However, these mainstream ciphers and functions are too large to fit in many types of embedded systems. Lightweight cryptography (LWC) focuses in designing cryptographic primitives for resource constraint devices. The main design goals in software are the reduction of processing and memory requirements. Embedded software implementations are optimized for throughput as well as memory and power savings.

Lightweight primitives provide moderate levels of security from 80 to 128 bits [41]. 80 bit security is adequate for constrained devices [42], like RFID tags and micro-controllers, while 128 bits is typical for mainstream applications [32].

In recent years, a high variety of lightweight proposals are presented [16]. The standardized primitives for LWC are referred in the ISO/IEC standard 29192 [43]. The part 2 of the standard includes block ciphers and the upcoming part 5 includes hash functions.

PRESENT [41] and CLEFIA [44] are the standardized block ciphers. PRESENT is a milestone in LWC due to its compact hardware implementation. CLEFIA was designed by SONY and is highly efficient both in hardware and software. In software, CLEFIA performs better than PRESENT [45].

PHOTON [46] and Spongent [47] are lightweight hash functions with sponge construction. They are considered for inclusion in the ISO/IEC 29192 standard for lightweight hash functions that is currently under development. Spongent is hardware oriented. PHOTON performs well both in embedded hardware and software.

The proposed LightPolyPHS scheme utilizes CLEFIA for cryptographic operations and PHOTON for hashing.

3 Lightweight PHS and PPH

LightPolyPHS is a lightweight PHS and PPH, designed for embedded systems and constrained devices. The overall system complies with the principles of LWC. First, we replace the inner cryptographic primitives that are utilized by the PHS Catena and the PPH PolyPassHash and implement two relevant lightweight schemes. Then, we integrate them by using the lightweight Catena as the PHS for the lightweight PolyPassHash and implement the proposed LightPolyPHS.

3.1 Catena PHS

Catena is suitable for multiple environments, like multi-core CPUs, databases, and low-memory devices. It is a composed cryptographic operation based on a cryptographic hash function and is simple and easy to analyse. The design constitutes a graph-based structure, called "Bit-Reversal Graph", that is instantiated by the cryptographic hash function. Any strong hash function can be embodied. The reference implementation selects SHA512 and BLAKE2b. SHA512 is standardized and widely-implemented in many platforms. BLAKE2b supports the Simple Instruction Multiple Data (SIMD) approach and protects massively parallel attacks on GPUs.

The scheme is well-documented with thorough security analysis. The time-memory tradeoff analysis is based on the pebble-game approach [10]. Catena provides lower bounds on the time-memory tradeoff, preimage security, indistinguishability from random and resistance against side-channel (e.g. cache-timing attacks [9]). The computational cost for massively parallel crackers on GPUs, ASICs and FPGAs is high.

HUIU is implemented by increasing the *t_cost* and *m_cost* parameters. For SR, the client can compute most of the PHS iterations while the server computes only the last one. Catena can also operate in a keyed password hashing mode by XORing the unkeyed output with the hash of the user ID, the *m_cost* and the secret key.

Lightweight Catena. Catena utilizes a cryptographic hash function to instantiate the graph-based structure. The reference implementation propose the functions SHA512 and BLAKE2b. Both hashes result in a 512-bit output and offer high level of security. SHA512 is selected as a widely-used and implemented standard while BLAKE2b is proposed due to its high efficiency and security against massively parallel attacks.

The lightweight Catena utilizes PHOTON-256 as the cryptographic hash function, which outputs a 256-bit digest. This results in a smaller datapath and implementation size than the original scheme as well as lower computational and memory requirements. The output size complies with the relevant primitive in PolyPassHash and provides moderate level of security. Table 1 summarizes the security properties of the three hash functions.

The security level of the Catena is determined by the underlying hash function. Consider that Catena-sha512, Catena-blake2b and Catena-photon256 offer 2^{512}, 2^{481} and 2^{244} bits security respectively.

3.2 PolyPassHash PPH

PolyPassHash is a PPH scheme that provides protection above PHS. It is composed of two building blocks: the aforementioned SSS threshold scheme and the

Table 1. Security properties of the examined hash functions

Hash function	**Hash Size (bits)**	**Collision Resistance**	**Pre-image Resistance**
SHA512[37]	512	2^{256}	2^{512}
BLAKE2[48]	512	2^{224}	2^{481}
PHOTON-256[46]	256	2^{128}	2^{244}

standardized SHA256 [37] hash function. The computational complexity of SSS is based on the k degree polynomial over a finite field. For PolyPassHash, the default k value is 3 and it is assigned as the *t_cost* parameter. SHA256 simple parses the password and the salt. The hashes are also encrypted with the AES.

At the server-side, PolyPassHash processes the password file when the system restarts. Then, a threshold of users must login before the passwords can be verified. After startup, the login requests are processed with similar computational overhead as in PHS-only systems. The memory overhead is about 1KB independent of the number of passwords and the storage cost is one byte per user account (the share value). An alternative partial verification process is also supported that allows users to login immediately after the restart without the need to verify a threshold of users.

No modification of the client applications or the login process is required. PolyPassHash is solely based on software and the system administrator can adjust the threshold value without affecting the users.

The attacker must guess 3 passwords simultaneously. On GPUs, PolyPassHash imposes about 23 orders of magnitude more effort than on PHS-only systems. On CPUs, even a threshold of 2 secrets provides sufficient security.

Lightweight PolyPassHash. PolyPassHash processes the passwords with SHA256. The hash function outputs 256-bits and AES with 128-bit key encrypts the SSS shares.

The lightweight PolyPassHash replaces SHA256 with the lightweight PHOTON256. AES is substituted by CLEFIA with the same key size. The two schemes exhibit the same datapath size and the resource saving is low. The security level of the lightweight version is similar to the original one.

In both schemes, the disk space requires 1 additional byte for each account to store the share value, in contrast to PHS-only solutions. The server must also store the polynomial coefficients for the SSS in memory. The total size is small: the XORed share and hash (256 bits long) multiplied by the threshold value (usually 2-5). In real systems, this value would result in a few hundred bytes.

3.3 PolyPHS

In PolyPassHash, passwords are simply parsed by SHA256. To further increase security, we replace the hash function with the Catena PHS. The PHS enhances resistance against attacks but is more resource demanding than SHA256. Also, Catena exhibits larger output size (512-bits) and the integrated implementation size is higher.

LightPolyPHS. The original Catena offers high level of security but it can not be applied on constrained devices. The lightweight Catena offers moderate level of security and is appropriate for the targeted systems. To fill the gap, the lightweight PolyPassHash is applied to increase security. The simple hash function is replaced by the PHS. Lightweight Catena uses the same datapath size as the SHA256 of PolyPassHash and provides higher password protection. With 3 shares as the threshold, an attacker must guess 3 lightweight-Catena passwords simultaneously to recover the password file. The security level is increased by 23 magnitudes on GPUs, resulting in $2^{244} * 10^{23} \approx 2^{320}$ bits security.

4 Evaluation

The examined PHSs, PPHs, and the core cryptographic primitives are evaluated under an Intel Core i7 at 2.10GHz CPU with 8GB RAM, running 64-bit operating systems. Reference C or C++ implementations are utilized in order to provide a fair comparison with the unoptimized versions of PHC [31]. All implementations are installed on Windows 8.1 Pro and are executed on cygwin. The different primitives are assessed under common assumptions. We measure the code size, memory consumption, execution time and throughput of each scheme.

4.1 Cryptographic primitives

Table 2 summarizes the software details of the block ciphers and hash functions that are examined in this study. The measured parameters are calculated from reference implementations. All lightweight primitives consume lower resources than the mainstream ones but are slower. For block ciphers, CLEFIA produces about 3.6 times smaller implementation than AES for slightly lower memory consumption and 1/12 of the speed. PRESENT has even lower code requirements but consumes more memory than AES and is slower than CLEFIA. For the hash functions, PHOTON consumes the least memory and exhibits the largest code size. Compared to SPONGENT it is about 8 times faster. Both CLEFIA and PHOTON can fit in constraint devices. The implementation size of both primitives is 21.9KB and the maximum RAM consumption can not exceed the 11.28KB.

4.2 PHSs and PPHs

Table 3 summarizes the software evaluation of the examined PHSs and PPHs based on the default sizes for output, password and salt, and the indicative *t_cost* and *m_cost* parameters as reported by each scheme.

Table 2. Software implementations of block ciphers and hash functions

Primitive	**Key/Hash Size (bits)**	**ROM (KB - lower is better)**	**RAM (KB - lower is better)**	**Throughput (MBps - higher is better)**
(Block ciphers)				
AES	128	20.0	10.25	56.35
PRESENT	128	2.6	14.5	0.44
CLEFIA	128	6.9	10.08	4.65
(Hash functions)				
SHA3	256	12	3.8	62.71
SHA3	512	12	3.9	36.80
SHA2	256	14	2.2	95.44
SHA2	512	14	2.4	36.91
BLAKE2b	256	14	2.18	96.66
BLAKE2b	512	14	2.38	42.82
Spongent	256	8.7	1.6	0.99
PHOTON	256	15	1.2	8.2

The standardized PBKDF2 is not memory-hard and consumes neglected memory. bcrypt has low memory requirements and achieves similar performance as scrypt. scrypt is the first widely-used memory-hard PHS and exhibits the higher memory consumption and larger implementation size.

Catena is a novel PHS that applies memory hardness to enhance security. Three versions are evaluated based on the underlying hash function. Catena-blake2d is the fastest and consumes similar memory as Catena-sha512. Catena-photon256 reduces memory demandings around 50% in exchange of lower performance. All three versions produce similar code size.

PolyPassHash is a novel PPH that utilizes the hash function SHA256 and the block cipher AES. It is quite efficient and has low and constant memory requirements. The Light-PolyPassHash version uses the hash function PHOTON and the block cipher CLEFIA. It decreases the code size and accomplishes slightly lower memory consumption and worsen speed.

The security of the initial scheme is fortified by replacing the hash function with a PHS. The PHS constitutes the most resource demanding component. The t_cost parameter determines the k shares of the SSS component and linearly affects the execution time. As t_cost increases, the number of password hashing operations, which are performed by the PHS, also increases. PolyPHS uses the Catena-blake2b ($t\_cost = 3$, $m\_cost = 18$) as the PHS of PolyPassHash. Light-PolyPHS uses the Catena-photon256 ($t\_cost = 3$, $m\_cost = 18$) as the PHS of Light-PolyPassHash.

Figure 3, illustrates the evaluation results of the 10 PHS and PPH schemes. For $k = 2$, LightPolyPHS has slightly worsen performance than bcrypt and scrypt. The memory-hard Catena-photon256 component enhanced with the SSS provide adequate security for around 39 times lower memory consumption and 2.3 smaller implementation size than scrypt.

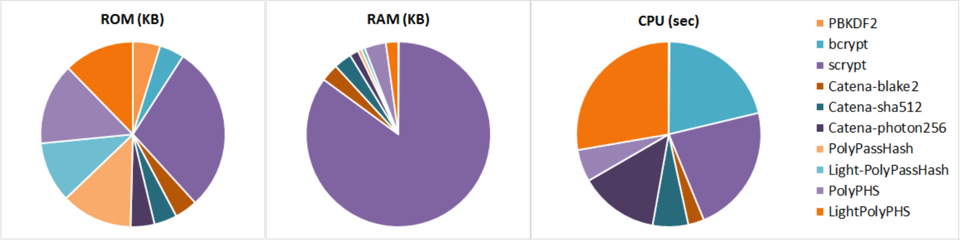


Fig. 3. Comparison of the examined PHSs

PHS	Password	Salt	Output	t_cost	m_cost	ROM	RAM	CPU(secs)
	(bytes)	(bytes)	(bytes)			(KB)	(KB)	
(PHSs)								
PBKDF2	24	8	64	1000	0	30	0	0.002024
	24	8	64	2048	0	30	0	0.004150
	24	8	64	4096	0	30	0	0.008141
	24	8	64	10000	0	30	0	0.019386
bcrypt	12	16	54	12	0	27	492	2.668653
scrypt	8	32	64	5	0	182	450656	2.837654
Catena–blake2b	8	16	64	3	18	25	16384	0.353742
	8	16	64	3	20	25	65596	2.619238
	8	16	64	3	21	25	128484	5.461030
Catena–sha512	8	16	64	3	18	25	16496	0.783590
	8	16	64	3	20	25	65720	5.389355
	8	16	64	3	21	25	131240	11.664960
Catena–photon256	8	16	32	3	18	26	8188	1.749200
	8	16	32	3	20	26	32760	13.065627
	8	16	32	3	21	26	65532	27.301944
(PPHs)								
PolyPassHash	16	16	32	1	0	78	3412	0.000054
	16	16	32	2	0	78	3412	0.000055
	16	16	32	4	0	78	3412	0.000055
Light–PolyPassHash	16	16	32	1	0	66	3410	0.000060
	16	16	32	2	0	66	3410	0.000068
	16	16	32	4	0	66	3410	0.000080
PolyPHS	16	16	64	1	0	89	19794	0.353695
	16	16	64	2	0	89	19794	0.707538
	16	16	64	4	0	89	19794	1.415020
LightPolyPHS	16	16	32	1	0	77	11579	1.749253
	16	16	32	2	0	77	11579	3.498454
	16	16	32	4	0	77	11579	6.996854

Table 3: Software implementations of PHSs and PPHs

4.3 MANET application

As a proof of concept, we apply LightPolyPHS on BeagleBone embedded devices [49]. BeagleBone is a credit-card-sized Linux computer with Internet connection that runs Android and Ubuntu OSs. The processor is an AM335x 720MHz ARM Cortex-A8 with 256MB DDR2 RAM and 4GB microSD.

We create a mobile ad hoc network (MANET) with 20 BeagleBone client devices and 1 BeagleBone acting as the server. The devices sense environmental parameters (e.g. temperature and moisture) and periodically upload this information to the server. They communicate wirelessly through USB-WiFi equipment in order to login the service and exchange data.

For 3 shares as the threshold, the server must authenticate three clients at initialization. Then, the account verification requires a single lightweight-Catena operation. The BeagleBone server takes on average 5 sec to startup and 1.8 sec to execute the PHS verification.

5 Conclusions

The maintenance of user passwords constitutes a significant factor related to the provided security of a service. Security breaches on famous applications have reveal massive amounts of user data, harming the reliability of their providers. The poor password hashing techniques and the limited available solutions lead the international cryptographic community to organize the Password Hashing Competition (PHC). The competition intends to delivery a small portfolio of modern and secure schemes for password hashing and key deviation. This paper presents the LightPolyPHS - a lightweight poly password hashing scheme for embedded systems and lightweight cryptography. We apply the proposed system on a MANET with BeagleBone embedded devices and held a comparative analysis with similar schemes on a mainstream computer. LightPolyPHS is the first lightweight password hashing and poly password hashing scheme suitable for constrained devices. Compared to current solutions it requires around 39 times less memory and 2.3 times smaller code size.

References

1. Richmond, S., Williams, C.: Millions of internet users hit by massive Sony PlayStation data theft, The Telegraph, London, April 26, 2011. http://www.telegraph.co.uk/technology/news/8475728/Millions-of-internet-users-hit-by-massive-Sony-PlayStation-data-theft.html (2011)
2. Finkle, J., Saba J.: LinkedIn suffers data breach - security experts, Reuters, June 2012. http://in.reuters.com/article/2012/06/06/linkedin-breach-idINDEE8550EN20120606 (2012)
3. Florencio, D., Herley, C., Van Oorschot, P. C.: An administrator's guide to internet password research, 28^{th} USENIX conference on Large Installation System Administration (LISA'14), Seattle, WA, November 9-14, 2014, pp. 35-52 (2014)
4. Telang, R., Wattal, S.: An Empirical Analysis of the Impact of Software Vulnerability Announcements on Firm Stock Price, IEEE Transactions on Software Engineering (TSE), IEEE, vol. 33, issue 8, 2007, pp. 544-557 (2007)
5. Kaliski, B., RSA Laboratories: RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0. Technical report, IETF, 2000 (2000)
6. Provos, N., Mazires, D.: A Future-Adaptable Password Scheme, USENIX Annual Technical Conference, pp. 8192 (1999)
7. Colin Percival. Stronger Key Derivation via Sequential Memory-Hard Functions. presented at BSDCan09, May 2009 (2009)

8. Orman, H: Twelve Random Characters: Passwords in the Era of Massive Parallelism, IEEE Internet Computing, vol. 17, issue 5, September-October, 2013, pp. 91-94 (2013)
9. Forler, C., Lucks, S., Wenzel, J.: Catena: A memory-consuming password scrambler, Cryptology ePrint Archive, Report 2013/525 (2013)
10. Forler, C., Lucks, S., Wenzel, J.: The Catena Password Scrambler, PHC submission, May 15, 2014. https://password-hashing.net/submissions/specs/Catena-v3.pdf (2014)
11. Cryptographic competition: Password Hashing Competition (PHC), April 25, 2013. https://password-hashing.net/ (2013)
12. Hatzivasilis, G., Papaefstathiou, I., Manifavas, C.: Password Hashing Competition – Survey and Benchmark, Cryptology ePrint Archive, Report 2015/265 (2015)
13. U.S. Department of Commerce, National Institute of Standards and Technology (NIST). http://www.nist.gov/
14. Seo, S.-H., Kim, J.-H., Hwang, S.-H., Kwon, K. H., Jeon, J. W.: A reliable gateway for in-vehicle networks based on LIN, CAN, and FlexRay, ACM Transactions on Embedded Computing Systems (TECS), vol. 11, issue 1, March 2012, Article No. 7 (2012)
15. Feng, A., Knieser, M., Rizkalla, M., King, B., Salama, P., Bowen, F.: Embedded system for sensor communication and security, IET Informaiton Security, vol. 6, issue 2, June 2012, pp. 111-121 (2012)
16. Manifavas, C., Hatzivasilis, G., Fysarakis, K., Rantos, K.: Lightweight cryptography for embedded systems a comparative analysis, 6th International Workshop on Autonomous and Spontaneous Security (SETOP 2013), ESORICS, Springer, LNCS, vol. 8247, 12-13 September, 2013, pp. 333-349. (2013)
17. Leu, J.-S., Hsieh, W.-B.: Efficient and secure dynamic ID-based remote user authentication scheme for distributed systems using smart cards, IET Informaiton Security, vol. 8, issue 2, March 2014, pp. 104-113 (2014)
18. Dhurjati, D., Kowshik, S., Adve, V., Lattner, C.: Memory safety without garbage collection for embedded applications, ACM Transactions on Embedded Computing Systems (TECS), vol. 4, issue 1, February 2005, pp. 73-111 (2005)
19. Bonneau, J., Schechter, S.: Towards reliable storage of 56-bit secrets in human memory, 23rd USENIX conference on Security Symposium (SEC'14), San Diego, CA, USA, 2014, pp. 607-623 (2014)
20. Li, C.: A new password authentication and user anonymity scheme based on elliptic curve cryptography and smart card, IET Information Security, vol. 7, issue 1, March 2013, pp. 3-10 (2013)
21. Sun, H.-M., Chen, Y.-H., Lin, Y.-H.: oPass: A User Authentication Protocol Resistant to Password Stealing and Password Reuse Attacks, IEEE Transactions on Information Forensics and Security, vol. 7, issue 2, 29 September, 2011, pp. 651-663 (2011)
22. NIST: Recommendation for Password-Based Key Derivation, NIST Special Publication 800-132, December 2010. http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf (2010)
23. Rajamanickam, V., Veerappan, D.: Inter cluster communication and rekeying technique for multicast security in mobile ad hoc networks, IET Information Security, vol. 8, issue 4, July 2014, pp. 234-239 (2014)
24. Lakshmi, R. P., Kumar, A. V. A.: Parallel key management scheme for mobile ad hoc network based on traffic mining, IET Information Security, vol. 9, issue 1, January 2015, pp. 14-23 (2015)

25. Van Oorschot, P. C., Thorpe, J.: On predictive models and user-drawn graphical passwords, ACM Transactions on Information and System Security (TISSEC), vol. 10, issue 4, January 2008, Article No. 5 (2008)
26. Shamir, A.: How to share a secret, Communications of the ACM, vol. 22, no. 11, 1979, pp. 612-613 (1979)
27. RSA Laboratories: PKCS #5: Password-Based Cryptographic Standard, version 2.0, 2000. http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-5-password-based-cryptography-standard.htm (2000)
28. Schneier, B.: Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), Fast Software Encryption, Springer, LNCS, vol. 809, 1994, pp. 191-204 (1994)
29. Percival, C., Josefsson, S.: The scrypt Password-Based Key Derivation Function, IETF, January 26, 2015. https://tools.ietf.org/html/draft-josefsson-scrypt-kdf-02 (2015)
30. Bernstein, D.J.: The Salsa20 family of stream ciphers, eSTREAM project, 2007. http://cr.yp.to/papers.html#salsafamily (2007)
31. Password Hashing Competition (PHC): Candidates, March 31, 2014. https://password-hashing.net/candidates.html (2014)
32. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: a Very Compact and a Threshold Implementation of AES. In: Advances in Cryptology EUROCRYPT 2011 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, vol. 6632, pp. 69 (2011)
33. Cakiroglu, M.: Software Implementation and Performance Comparison of Popular Block Ciphers on 8-bit Low-Cost Microcontroller. International Journal of the Physical Sciences, vol. 5, issue 9, pp. 1338-1343, 18 August (2010)
34. Eisenbarth, T., Gong, Z., Guneysu, T., Heyse, S., Indesteege, S., Kerckhof, S., Koeune, F., Nad, T., Plos, T., Regazzoni, F., Standaert, F.-X., Oldenzeel, Loic van Oldeneel: Compact Implementation and Performance Evaluation of Block Ciphers in ATiny Devices. ECRYPT Workshop on Lightweight Cryptography, Louvain-la-Neuve, Belgium (November 2011), and AFRICACRYPT 2012, LNCS, vol. 7374, pp. 172-187. Springer (2012)
35. Feldhofe, M., Rechberger, C.: A Case Against Currently Used Hash Functions in RFID Protocols. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops, LNCS, vol. 4277, pp. 372-381. Springer, Heidelberg (2006)
36. NIST: Secure Hash Standard (SHS), FIPS PUB 180-4, March 2012. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf (2012)
37. NIST: Secure Hash Standard, FIPS 180-2, April 1995. http://csrc.nist.gov (1995).
38. Sugawara, T., Homma, N., Aoki, T., Satoh, A.: Compact ASIC Architectures for the 512-Bit Hash Function Whirlpool. In: Chung, K.-I., Sohn, K., Yung, M. (eds.) Information Security Applications. LNCS, vol. 5379, pp 28-40. Springer, Berlin, Heidelberg (2009)
39. SHA3 Contest, NIST, http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html
40. Aumasson, J.-P., Neves, S., Wilcox-OHearn, Z., Winnerlein, C.: BLAKE2: Simpler, Smaller, Fast as MD5, ACNS, Springer, LNCS, vol. 7954, 2013, pp 119135 (2013)
41. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Proceedings of Workshop Cryptographic Hardware and Embedded Systems (CHES 07) (2007)

42. Shibutani, K., Isobe, T., Hiwarati, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. Cryptographic Hardware and Embedded Systems CHES 2011, LNCS, vol. 6917/2011, pp. 342-357. Springer (2011)
43. ISO/IEC 29192:2012, International standard for lightweight cryptographic methods, ISO/IEC, 2012, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=56425
44. Akishita, T., Hiwatari, H.: Very Compact Hardware Implementations of the Blockcipher CLEFIA, Sony Corporation, Technical Paper, June 2011, http://www.sony.co.jp/Products/cryptography/clefia/download/data/clefia-hwcompact-20110615.pdf (2011)
45. Hatzivasilis, G., Theodoridis, A., Gasparis, E., Manifavas, C.: ULCL: an Ultra-Lightweight Cryptographic Library for embedded systems, Measurable security for Embedded Computing and Communication Systems (MeSeCCS 2014), within the 4th International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2014), 7-9 January, 2014, Lisbon, Portugal (2014)
46. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011, LNCS, vol. 6841, pp. 222-239 (2011)
47. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011, LNCS, vol. 6917, pp. 312-325 (2011)
48. Guo, J., Karpman, P., Nikolic, I., Wang, L., Wu, S.: Analysis of Blake2, Cryptology ePrint Archive, Report 2013/467 (2013)
49. BeagleBoard.org Foundation: BeagleBone manual, 2011, http://beagleboard.org/bone (2011)