Energy Consumption of Hash Functions

R. Damasevicius[1], G. Ziberkas[1], V. Stuikys[1], J. Toldinas[2]
[1]*Software Engineering Department, Kaunas University of Technology, Studentu St. 50, LT-51368, Kaunas, Lithuania, phone: +370 37 300399*
[2]*Computer Department, Kaunas University of Technology, Studentu St. 50, LT-51368, Kaunas, Lithuania, phone: +370 37 300389*
eugenijus.toldinas@ktu.lt

***Abstract*—We analyse energy efficiency versus quality characteristics of hashing algorithms in a mobile device and describe methodologies for energy measurement on a Java-enabled smart phone. Energy efficiency of 17 hash functions (Adler32, Crc16, Crc32, Haval256, MD2, MD4, MD5, MD6, SHA1, SHA224, SHA256, SHA384, SHA512, Skein, SV1, Tiger, Whirlpool) is evaluated using the GSM modem-based battery charge measurement method, and quality is evaluated using the Avalanche and Chi-square tests. The results show that the most energy-efficient hash function on a mobile device is SV1 for cryptographic applications, and crc16 for non-cryptographic applications.**

***Index Terms*—Energy consumption, algorithms, memory management, encryption, authentication.**

I. INTRODUCTION

Energy efficiency is extremely important in mobile devices (smart phones, lap tops, tablet PCs, handhelds, etc.) that operate using limited battery power. Currently, the growth of complexity of functions provided by mobile devices outpaces evolution of the battery technologies [1]. In order to have acceptable battery life, mobile devices must stay within a fixed energy budget. The increasing demand for mobile services drives scientific research into finding architectural and computational solutions to overcome this limitation [2]. When developing mobile applications and providing wireless communication services, security and reliability of transmitted messages or data stored on a mobile device is of paramount importance. A wide class of functions, called hash functions, is commonly used in cryptographic and error control services to provide message authentication and integrity.

In this paper, we: 1) analyze the hash function domain; 2) describe a methodology for measuring energy efficiency of hash functions on a smart phone; 3) describe the experimental results of energy consumptions vs. quality of hash functions; and 4) provide recommendations for mobile application developers.

II. ANALYSIS OF HASH FUNCTION DOMAIN

A hash function is a deterministic procedure that converts a large, possibly variable-sized amount of data (message) into a small fixed-size block of data (hash value). The aim is to detect accidental or intentional changes to data, or to determine whether or not two pieces of data are identical. Hash functions are primarily used in memory management (hash tables), cryptographic applications (message authentication and encryption), and error detection and control (data integrity checking), though there are other applications in computer graphics, computational geometry, sorting, searching, etc. The following classes of hash functions can be distinguished:

Checksum algorithms such as CRC32 and other cyclic redundancy checks are error-detecting codes designed to detect accidental changes to raw data. CRC32 was previously used for message integrity in the WEP (*Wired Equivalent Privacy*) encryption IEEE 802.11 wireless networks, however, due to weak security it is now considered to be unsuitable for cryptographic applications.

Message digest algorithms such as MD5 are functions commonly used to check data integrity. Digests are widely used to provide some assurance that a transferred file has arrived intact, and for computing digital signatures of documents before encryption.

Cryptographic hash functions such as SHA512 are hash functions that must be able to withstand all known types of cryptanalytic attack and have very strong cryptographic requirements [3].

Perceptual hashes are fingerprints designed for content recognition and identification of copyrighted content such as images, audio, video clips.

Good hash functions are required to satisfy certain properties such as low computability cost, determinism, spread, randomness, regularity and uniformity [4]. Here determinism means that the hash value is fully determined by the data being hashed. Spread means that the hash function generates very different hash values for similar data. Randomness means that it is computationally difficult to find two different messages that have equal hash values. Uniformity means that the distribution of hash values obtained from a set of non-uniform data should be statistically distinguishable from the uniform distribution.

A cryptographic hash function must satisfy additional properties in order to be able to withstand all known types of cryptanalytic attack: 1) Preimage resistance: given a hash value it should be computationally difficult to find any message that has a given hash value. 2) Second preimage

Manuscript received March 19, 2012; accepted April 26, 2012.

resistance: given a message it should be difficult to find another message that has the same hash value.

Therefore, a good-quality hash function should behave as much as possible like a random number generator while still being deterministic and efficiently computable.

When testing the quality of a hash function, statistical tests (Avalanche test, Collision test, Birthday Attack, etc.) can be applied [5].

Uniformity of the distribution of hash values can be evaluated by the χ^2 test. It tests a null hypothesis stating that the frequency distribution of certain events observed in a sample is consistent with a particular theoretical distribution

$$\chi^2 = \sum_{i=1}^{M} n_i \frac{(n_i - P)^2}{P}, \quad (1)$$

where M is the number of samples (bins), n_i is the number of hash values in i-th sample, and P is the theoretical distribution value.

Spread of hash values can be evaluated using an Avalanche test. A hash function has an Avalanche effect if the Hamming distance between the outputs of a random input vector and one generated by randomly flipping one of the input bits should be, on average, equal to $n/2$, where n is the length of the output. The test is evaluated using the χ^2 statistic that measures the distance of the observed distribution of the Hamming distances from the theoretical Bernoulli probability distribution $B(1/2, n)$.

Regularity of a hash function can be quantified using the Birthday Attack. The method used to find a collision is to evaluate the hash function h for different randomly chosen input values until the same result is found more than once. If the outputs of the function are distributed unevenly, then a collision can be found even faster. Regularity of a hash function quantifies the resistance of the function to Birthday attacks and allows its vulnerability to be estimated [6] as

$$\mu = \frac{1}{2}\log_R Q\ . \quad (2)$$

where R is the range of a hash function, and Q is the average number of hash function tests until a first collision is found. Value of less than 1 means higher vulnerability.

III. ENERGY MEASUREMENT METHODOLOGIES

Additionally to low computability criterion, we emphasize the importance of low energy consumption of hash functions. To measure energy consumption on a Java-enabled smart phone, the following measurement methodologies can be used:

1) *Java-based*: battery charge is measured by reading the Java system property "batterylevel" (different phones may have differing names of this property) during execution of Java application. Snippet of code in Fig. 1 returns the current battery level (0-100).

2) *Sensor-based:* battery charge is measured by reading sensor values that return the battery charge. This works only for devices that support JSR 256 Mobile Sensor API. Snippet of code in Fig. 2 returns the current battery charge level (0-100).

```
String bLevel =
   System.getProperty("batterylevel");
if (bLevel != null)
   form.append (bLevel);
else
   form.append ("Not supported");
```

Fig. 1. Battery charge level measurement in Java application using system property.

```
SensorManager sm;
SensorInfo[] bInfo =
    sm.findSensors("battery_charge", null);
SensorConnection sensor = (SensorConnection)
    Connector.open(batteryInfo[0].getUrl());
Data data[] = sensor.getData(1);
String bLevel = "Current charge level: " +
    data[0].getIntValues()[0];
```

Fig. 2. Battery charge level measurement in Java application using JSR 256 API.

3) *Modem-based:* battery charge is measured by connecting an external PC via USB to the GSM modem of a phone and issuing a specific command of the Hayes (AT) command language. The command "AT+CBC" returns a battery charge (0-100). The measurement procedure is as follows:

1) Enter the engineering mode on a phone;
2) Change the UART controller mode from USB to serial communication (COM);
3) Switch off and switch on the phone;
4) Connect the phone to a PC via USB;
5) Make sure that the phone's battery is not charging (wait until it is fully charged and stops charging until next reconnect of the cable, or cut wire 1 in the USB cable);
6) Run a mobile application, whose energy consumption is measured, on the phone;
7) Run a battery charge measurement script on the PC;
8) Wait until mobile application finishes and read the results on the PC.

The advantage of this methodology is that it is implementation-language-independent. A snippet of the measurement script implemented in a Perl script language using the Device::Modem module is presented in Fig. 3.

```
use Device::Modem;

$modem = new Device::Modem( port =>'COM6' );
$modem->connect( baudrate => 115200 );
$modem->send_init_string();
$modem->atsend( "AT+CBC\r" );
($ok,@result) = $modem->parse_answer();
$charge = substr(@result[0], 8);
print "Battery charge: ", $charge, "%\n";
```

Fig. 3. Snippet of battery charge measurement script in Perl.

IV. EXPERIMENTS

A number of freely available Java implementations of

hash functions were selected. Adler-32 is a checksum function that is a part of the widely-used *zlib* compression library. CRC16 and CRC32 are cyclical redundancy checks used in a variety of communication standards. MD2, MD4, MD5, MD6 are message digest algorithms. Haval256 is a cryptographic hash function that can be used as a replacement of MD5. SHA1, SHA224, SHA256, SHA384, SHA512 are cryptographic hash functions published by the National Institute of Standards and Technology (NIST, USA). Skein is a cryptographic hash function and one out of five finalists in the NIST hash function competition to replace the SHA-3 standard. SV1 is a cryptographically strong digest algorithm. Tiger is a cryptographic hash function designed for efficiency on 64-bit platforms. Whirlpool is a cryptographic hash function, which is a part of the ISO/IEC 10118-3 standard.

The experiments were performed using SciPhone i9+++ smart phone (a clone of iPhone), CPU MTK MT6225A or MT6318A 26 MHz, MTK OS (a version of Nucleus Plus OS), Java HotSpot VM 1GB heap memory, support for Jazelle extension, Li-ion battery 3.7 V, 1800 mAh.

Energy consumption of hash functions has been measured using the modem-based method, because the phone does not support the sensor-based and Java-based methods. Phone display was switched off to reduce system energy consumption. Hashing was performed on a randomly generated file of 100 MB size and repeated 100 times. As the method returns the percentage value of battery drain, the result was multiplied by the battery capacity value to obtain the energy consumption value in Watts. Net power consumption (per message MB) and energy consumption results are presented in Fig. 4 and Fig. 5, respectively.

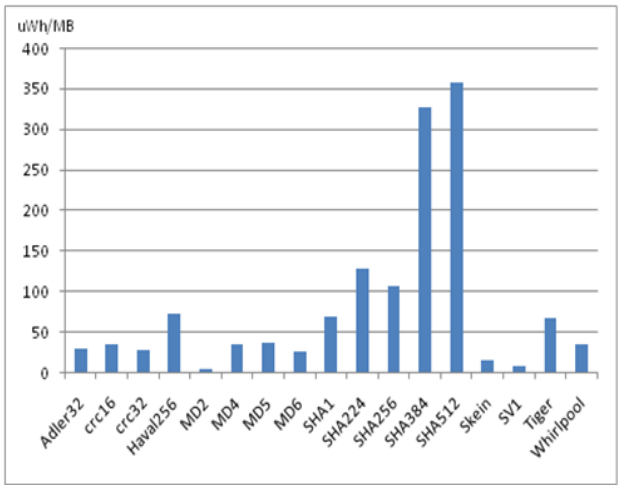

Fig. 4. Net power consumption (uWh/MB) of hash functions.

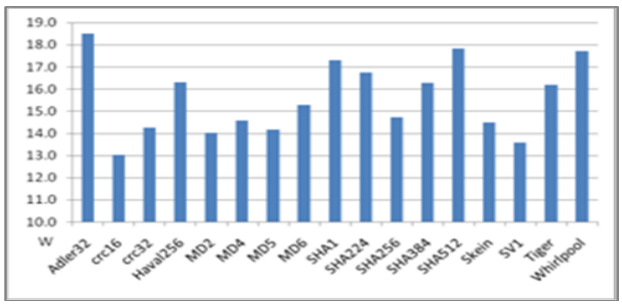

Fig. 5. Average energy consumption (W) of hash functions.

Hash function quality characteristics based on the results of the collision test, χ^2 test and avalanche test are presented in Table 1. The tests were performed on randomly generated 160-byte length string (max. payload of SMS messages), using 2^{10} bins and 2^{20} trials.

The Collision Test and Birthday Attack were not performed due to large computational requirements.

TABLE I. RESULTS OF χ^2 AND AVALANCHE TESTS.

Hash function	χ^2 test, χ^2	Avalanche test, χ^2
Adler32	1.028	4.714E-1
Crc16	1.003	2.286E-1
Crc32	1.005	4.144E-1
Haval256	1.040	2.938E-3
MD2	1.041	3.088E-3
MD4	1.041	2.079E-3
MD5	1.042	2.095E-3
MD6	1.043	3.824E-3
SHA1	1.042	2.432E-3
SHA224	1.042	3.447E-3
SHA256	1.041	2.767E-3
SHA384	1.041	4.222E-3
SHA512	1.039	4.426E-3
Skein	1.042	4.725E-3
SV1	1.041	1.618E-3
Tiger	0.769	3.275E-3
Whirl-pool	1.042	4.239E-3

The results of the χ^2 test were not conclusive for determining differences in hash function quality (all functions generate uniform hash values).

The trade-offs of power and energy consumption vs. quality of hash functions using Avalanche test results as a quality metric are presented in Fig. 6 and Fig. 7.

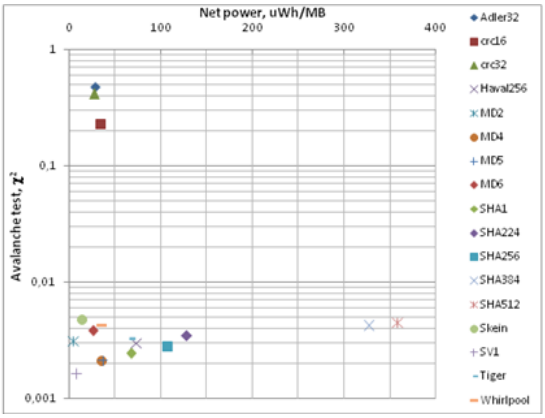

Fig. 6. Net power consumption vs. Avalanche test.

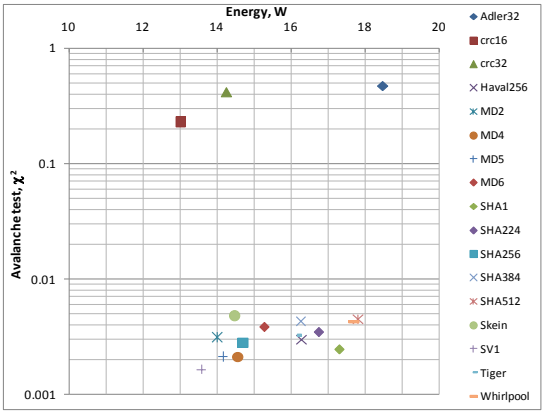

Fig. 7. Energy consumption vs. Avalanche test.

V. RELATED WORK

Energy efficiency of the cryptographic hash functions implemented as software has been addressed previously as a part of the studies of cryptographic algorithms and services however, there has not been wide scale surveys, yet. For example, Kaps *et al.* [7] analyse energy consumption of the WH hash function. Energy efficiency of SHA1, SHA2 and MD5 is analysed in [8].

VI. CONCLUSIONS

For experiments we use the sciPhone smart phone and Java ME as a modern and popular platform for safe development mobile applications and secure information management. Though there are many hash algorithms known, we were restricted with the algorithms provided by this framework. The energy-efficiency of hash algorithms with varying block and hash sizes is highly different.

The main results of this paper are as follows. The results show that the most energy-efficient hash function on a mobile device is *SV1* for cryptographic applications, and *crc16* for non-cryptographic applications. The energy-aware selection of hash function can save up to 29% (the difference between min. and max. consumption of energy among analyzed hash functions) energy on hashing operations.

REFERENCES

[1] F. C. C. Osorio, E. Agu, K. McKay, "Measuring energy-security tradeoffs in wireless networks", in *Proc. of the 24th IEEE Int. Performance Computing and Communications Conference (IPCCC 2005)*, Phoenix, Arizona, USA, IEEE, 2005, pp. 293–302. [Online]. Available: http://dx.doi.org/10.1109/PCCC.2005.1460572

[2] R. Damaševičius, V. Štuikys, E. Toldinas, "Embedded program specialization for multiple criteria trade–offs", *Elektronika ir Elektrotechnika (Electronics and Electrical Engineering)*, no. 8, pp. 9–14, 2008.

[3] S. Danker, R. Ayers, R. P. Mislan, "Hashing Techniques for Mobile Device Forensics", "Small Scale Digital Device Forensics Journal". vol. 1, no. 3, 2009.

[4] M.-J.O. Saarinen, "Cryptanalysis of Dedicated Cryptographic Hash Functions", Ph.D. dissertation, University of London, 2009.

[5] F. Sulak, A. Doganaksoy, B. Ege, O. Kocak, "Evaluation of randomness test results for short sequences", in *Proc. of the 6th Int. Conf. on Sequences and their applications (SETA'10)*, Springer-Verlag, 2010, pp. 309–319.

[6] M. Bellare, T. Kohno, "Hash Function Balance and Its Impact on Birthday Attacks", in *Proc. of the Int. Conf. on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2004)*, Interlaken, Switzerland, LNCS, vol. 3027, 2004, pp. 401–418.

[7] J.-P. Kaps, K. Yuksel, B. Sunar, "Energy Scalable Universal Hashing", *IEEE Transactions on Computers*, vol. 12, no. 54, pp. 1484–1495, 2005. [Online]. Available: http://dx.doi.org/10.1109/TC.2005.195

[8] X. Ruan, A. Manzanares, S. Yin, M. Nijim, X. Qin, "Can We Improve Energy Efficiency of Secure Disk Systems without Modifying Security Mechanisms?", in *Proc. of the 2009 IEEE Int. Conf. on Networking, Architecture, and Storage (NAS 09)*, IEEE Computer Society, 2009, pp. 413–420.