Generation of the Secret Encryption Key Using the Signature of the Embedded System

Algimantas Venčkauskas
Computer Department, Kaunas University of Technology,
Studentų str. 50-212, LT-51368, Kaunas, Lithuania
e-mail: algimantas.venckauskas@ktu.lt

Nerijus Jusas
Computer Department, Kaunas University of Technology,
Studentų str. 50-210, LT-51368, Kaunas, Lithuania
e-mail: nerijus.jusas@ktu.lt

Irena Mikuckienė
System Analysis Department, Kaunas University of Technology,
Studentų str. 50-211, LT-51368, Kaunas, Lithuania
e-mail: irena.mikuckiene@ktu.lt

Stasys Maciulevičius
Computer Department, Kaunas University of Technology,
Studentų str. 50-209, LT-51368, Kaunas, Lithuania
e-mail: stasys.maciuveicius@ktu.lt

crossref http://dx.doi.org/10.5755/j01.itc.41.4.1162

Abstract. Program protection, programming code integrity and intellectual property protection are important problems in embedded systems. Security mechanisms for embedded systems have some specific restrictions related to limited resources, bandwidth requirements and security. In this paper we develop a secret encryption key generation algorithm by using the signature of the embedded system. We explore the qualitative characteristic of the generated keys - the entropy. Experiments showed that the generated secret keys have high entropy.

Keywords: embedded system, program protection, secret encryption key, hash function, entropy.

1. Introduction

It is hard to imagine most of today’s appliances and devices, electronics, telecommunications, mechatronics, etc., without using the embedded systems. These systems face significant challenges in information security; on the one hand, they usually have very limited resources and, on the other hand, they function in a physically unsafe environment. The embedded systems usually perform critical functions: controlling important real time objects and processing important information. Therefore, their work is open to sabotage.

Security requirements for the embedded systems depend on specific areas of application [16, 26]. The following requirements are related to the general requirements for information security: integrity, availability and confidentiality. However, the specificity of the embedded systems, their mobility and operation in real time, typically have certain limitations such as processing gap, energy gap, flexibility, tamper resistance, assurance gap and cost. This is largely due to limited resources, performance and security requirements.

An important component of the embedded system that also influences its performance and vitality is software. Software security has two aspects: secure program and program protection [19]. We will explore the protection aspect of program security. The main program protection vulnerabilities are the following...
Generation of the Secret Encryption Key Using the Signature of the Embedded System

[23]: violation of intellectual property (illegal copying and distribution, improper use of licenses, reverse engineering), disclosure of software code, theft of algorithms and falsification of software code.

According to the study by the Business Software Alliance (BSA) [4], the software creators lost USD 51.4 billion and the pirated software accounted for 43% of all software with piracy growing by around 2 percent annually.

No matter what threats software is protected from, for example copying or the theft of algorithms, the attackers use a wide range of means to crack the protection: reverse engineering, including disassembly and decompilation, debuggers, disassemblers, compilers, emulators, simulators and spoofing attacks [18].

There are many software protection methods, which are divided into software and hardware-based. Software-based protection mechanisms are integrated into the software or the algorithm, which is protected and can be added to the software code: code and data obfuscation [6], anti-debugging method [7], code encryption technology, self-modifying code and self-extracting code [13]. Hardware-based methods can significantly increase the level of protection, largely due to the fact that they are external devices in which the level of protection is controlled by the software provider and not by the end-user [12, 17, 20]. A part of the program code or data (encryption keys) required to run the program can be stored in the additional hardware (commonly Dongle or USB keys). However, this protection mechanism is relatively expensive and is generally only used for the programs which are of great commercial value.

Intermediate software/hardware methods are also used: tethering the program to a computer or device signatures (CPU, RAM, ROM, BIOS, OS, etc., serial numbers, model ID, etc.) [21, 25, 31]. Firewalls are also used for the protection of the internet programs [14]. These methods are usually used for anti-piracy in personal computers.

Gelbart et al. [8] proposed a joint compiler/hardware infrastructure for protection of the embedded system software for fully encrypted execution in which both program and data are in the memory in the encrypted form. The processor is supplemented with the Field Programmable Gate Array (FPGA)-based secure hardware component. Arora et al. [1] presented architecture for hardware-assisted run-time monitoring, wherein the embedded processor is augmented with the hardware monitor that observes the dynamic execution trace of the processor and checks whether the execution trace falls within the allowed program behaviour.

In assessing the limitations of the embedded systems [2], one of the most acceptable software protection methods is code encryption. However, key management faces additional issues: it requires an additional storage medium, the encryption keys have to be entered manually and key transfer via the network must be protected by using SSL protocol and others [27, 28].

Secret encryption keys are used for various purposes in the embedded systems, such as communication, data encryption, etc. Physical characteristics of the embedded system, such as physical unclonable functions (PUF) are used for the generation of keys [24, 29].

Our goal is to create a protection method of the embedded system software that does not require external hardware and infrastructure for key generation, storage and management and provides a sufficient level of security. The code of the embedded system software is stored in an encrypted form; secret encryption keys are generated in real time, on demand, before the execution of the encrypted software module.

In the following sections, we describe the proposed method of secret key generation by using the signature of the embedded system and investigate its characteristics and possibilities of its application for the protection of the embedded system software.

2. The method for generation of the secret key

The secret encryption key of the software module is generated from the headers of the program to be protected and from the signatures of the embedded system hardware and software components (CPU, RAM, ROM, BIOS, OS, etc.), using the fastest and simplest logical operations (XOR, OR, SHIFT). For convenience of description, terminology and notations used in the paper are summarized as follows:

- \( K = \{ k_i \} \): program encrypting key.
- \( P = \{ p_i \} \): header of the program to be protected. The program header is an array of structures, each describing a segment or other information the system needs to prepare the program for execution.
- \( PSN = \{ psn_i \} \): a serial number of the program to be protected.
- \( PH = \{ ph_i \} \): a hash of header of the program to be protected.
- \( SS = \{ ss_i \} \): signature of the embedded system.
- \( ES = \{ es_i \} \): signature of the embedded system components.
- \( CV = \{ cv_i \} \): component vendor identifiers (ID).
- \( CT = \{ ct_i \} \): component type ID.
- \( CM = \{ cm_i \} \): component model ID.
- \( CSN = \{ csn_i \} \): component serial number.
- \( \| \) : string concatenation operation.
- \( \lor \) : the bitwise OR operation.
- \( \oplus \) : the bitwise exclusive OR (XOR) operation.
- \( \mod n \) : modulo \( n \) operation.
- \( h(\cdot) \) : a cryptographic one-way hashing function (MD5, SHA-1…).
– $eb(s, k)$: function for the extraction of $k$ bytes from the string $s$.
– $sign()$: function for creating the signature of the embedded system (defined below).
– $key()$: function for generation of the secret encryption key (defined below).

The process of generation of the secret key consists of five steps (Fig. 1).

1. Create the set of signatures of the components of the embedded system $ES = \{es_i\}, i = 1, \ldots, n$. The signature is created by applying the string concatenation of Vendor ID ($cv_i$), Type ID ($ct_i$), Model ID ($cm_i$), and Serial Number ($csn_i$):

   $$ es_i = cv_i || ct_i || cm_i || csn_i. $$

In steps 2 – 6 a subset of the component signatures is created. These signatures will be used for computing the embedded system signature.

2. Calculate the program header hash $ph = h(p || psn)$.
3. Create the $n \times m$ matrix $MH = \{mh_{ij}\}$ from the bytes of the program header hash $mh_{ij} = eb(ph, (i - 1) \times j + i)$, where $n$ is the number of the embedded system signatures, and $m = eb(ph, n) \mod n$.
4. Calculate the sum $s_j$ of the column elements in the matrix $MH$, $s_j = \sum_{i=1}^{n} mh_{ij}, j = 1, \ldots, m$.
5. Create the index array of the component signatures $IND = \{ind_j\}$, where $ind_j = s_j \mod n$ and delete repetitive indices, $ind_j \neq ind_p, \forall i \in \{1 \ldots j - 1\}$.
6. Create the subset of the component signatures $\bar{ES} \subseteq ES, \bar{es}_j = es_j, \forall j = ind_p, \forall ind_j \in IND, k = 1, \ldots, m$, from which the embedded system signature will be created.
7. Create the signature of the embedded system $ss_i = sign(\bar{ES})$.
8. Generate the program protection key $key(ss, salt, iteration\_count, key\_length)$, where salt, iteration_count and key_length are defined below.

Now, we describe the set of $sign$ functions. The signature of the embedded system is created by processing byte strings of the component signatures. All $sign$ functions can be performed using bit operations (bitwise OR, bitwise XOR, bitwise AND, SHIFT).

**Function $sign1$**. Bitwise XOR is used to create a signature of the embedded system (Fig. 2).

```
Function $sign1$. Bitwise XOR is used to create a signature of the embedded system (Fig. 2).

input: ess, m // subset of m component signatures
output: ss // signature of the embedded system
l := maxlength ess // max of the length of the component signatures
for j = 1 to l do
    ss (j) := ess (1, j)
end for
for i = 2 to m do
    for j = 1 to l do
        ss (j) := ss (j) XOR ess (i, j)
    end for
end for
```

**Function $sign2$**. Bitwise OR is used to create a signature of the embedded system (Fig. 3).

```
Function $sign2$. Bitwise OR is used to create a signature of the embedded system (Fig. 3).

input: ess, m // subset of m component signatures
output: ss // signature of the embedded system
l := maxlength ess // max of the length of the component signatures
for j = 1 to l do
    ss (j) := ess (1, j)
end for
for i = 2 to m do
    for j = 1 to l do
        ss (j) := ss (j) OR ess (i, j)
    end for
end for
```

**Function $sign3$**. Bitwise OR and XOR are used to create a signature of the embedded system (every other component signature are processed by XOR and OR operation, starting with XOR, see Fig. 4).
Generation of the Secret Encryption Key Using the Signature of the Embedded System

In a practical way, source entropy is computed from the observed frequency for every character by means of the previous formula.

So, the entropy $S$ of the string is defined as follows:

$$S = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)),$$

where $n$ is the number of possible values of character, $m$ is the length of the string, and $\text{num}(x_i)$ is the number of appearances of character $x_i$ in the string.

The initial data of the experiment (the header of the program protection key is defined as follows:

$$k = \text{key}(ss, salt, \text{iteration\_count}, key\_length).$$

In the following section, we investigate the characteristics of the method for generation of the secret key by using the signature of the embedded system.

3. Evaluation of the proposed method

The encryption keys have to be generated truly randomly, to contain sufficient entropy and be of sufficient length [30]. Since our problem is the protection of the embedded system software, the key generation process must be carried out without any additional hardware and infrastructure cost [2, 26]. The proposed method generates the secret keys only from the signature of the embedded system. Further, we investigate whether the entropy of the key is high enough (close to 1). The maximum value of entropy of a bit string is 1 [30, 5, 3].

The concept of entropy is defined by Shannon. Let us consider an information source described as a sequence of instances of a random (discrete) variable $X$, which can take a finite number of possible values $x_1, x_2, \ldots, x_n$ with a probability respectively equal to $p_1, p_2, \ldots, p_n$ (in other words $p(x_i) = p_i$). Then the source entropy is defined by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)).$$

In a practical way, source entropy is computed from the observed frequency for every character by means of the previous formula.

So, the entropy $S$ of the string is defined as follows:

$$S = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)),$$

where $n$ is the number of possible values of character, $m$ is the length of the string, and $\text{num}(x_i)$ is the number of appearances of character $x_i$ in the string.

The initial data of the experiment (the header of the program protection key is defined as follows:

$$k = \text{key}(ss, salt, \text{iteration\_count}, key\_length).$$

In the following section, we investigate the characteristics of the method for generation of the secret key by using the signature of the embedded system.

3. Evaluation of the proposed method

The encryption keys have to be generated truly randomly, to contain sufficient entropy and be of sufficient length [30]. Since our problem is the protection of the embedded system software, the key generation process must be carried out without any additional hardware and infrastructure cost [2, 26]. The proposed method generates the secret keys only from the signature of the embedded system. Further, we investigate whether the entropy of the key is high enough (close to 1). The maximum value of entropy of a bit string is 1 [30, 5, 3].

The concept of entropy is defined by Shannon. Let us consider an information source described as a sequence of instances of a random (discrete) variable $X$, which can take a finite number of possible values $x_1, x_2, \ldots, x_n$ with a probability respectively equal to $p_1, p_2, \ldots, p_n$ (in other words $p(x_i) = p_i$). Then the source entropy is defined by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)).$$

In a practical way, source entropy is computed from the observed frequency for every character by means of the previous formula.

So, the entropy $S$ of the string is defined as follows:

$$S = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)),$$

where $n$ is the number of possible values of character, $m$ is the length of the string, and $\text{num}(x_i)$ is the number of appearances of character $x_i$ in the string.

The initial data of the experiment (the header of the program protection key is defined as follows:

$$k = \text{key}(ss, salt, \text{iteration\_count}, key\_length).$$

In the following section, we investigate the characteristics of the method for generation of the secret key by using the signature of the embedded system.

3. Evaluation of the proposed method

The encryption keys have to be generated truly randomly, to contain sufficient entropy and be of sufficient length [30]. Since our problem is the protection of the embedded system software, the key generation process must be carried out without any additional hardware and infrastructure cost [2, 26]. The proposed method generates the secret keys only from the signature of the embedded system. Further, we investigate whether the entropy of the key is high enough (close to 1). The maximum value of entropy of a bit string is 1 [30, 5, 3].

The concept of entropy is defined by Shannon. Let us consider an information source described as a sequence of instances of a random (discrete) variable $X$, which can take a finite number of possible values $x_1, x_2, \ldots, x_n$ with a probability respectively equal to $p_1, p_2, \ldots, p_n$ (in other words $p(x_i) = p_i$). Then the source entropy is defined by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)).$$

In a practical way, source entropy is computed from the observed frequency for every character by means of the previous formula.

So, the entropy $S$ of the string is defined as follows:

$$S = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)),$$

where $n$ is the number of possible values of character, $m$ is the length of the string, and $\text{num}(x_i)$ is the number of appearances of character $x_i$ in the string.

The initial data of the experiment (the header of the program protection key is defined as follows:

$$k = \text{key}(ss, salt, \text{iteration\_count}, key\_length).$$

In the following section, we investigate the characteristics of the method for generation of the secret key by using the signature of the embedded system.

3. Evaluation of the proposed method

The encryption keys have to be generated truly randomly, to contain sufficient entropy and be of sufficient length [30]. Since our problem is the protection of the embedded system software, the key generation process must be carried out without any additional hardware and infrastructure cost [2, 26]. The proposed method generates the secret keys only from the signature of the embedded system. Further, we investigate whether the entropy of the key is high enough (close to 1). The maximum value of entropy of a bit string is 1 [30, 5, 3].

The concept of entropy is defined by Shannon. Let us consider an information source described as a sequence of instances of a random (discrete) variable $X$, which can take a finite number of possible values $x_1, x_2, \ldots, x_n$ with a probability respectively equal to $p_1, p_2, \ldots, p_n$ (in other words $p(x_i) = p_i$). Then the source entropy is defined by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)).$$

In a practical way, source entropy is computed from the observed frequency for every character by means of the previous formula.

So, the entropy $S$ of the string is defined as follows:

$$S = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)),$$

where $n$ is the number of possible values of character, $m$ is the length of the string, and $\text{num}(x_i)$ is the number of appearances of character $x_i$ in the string.

The initial data of the experiment (the header of the program protection key is defined as follows:

$$k = \text{key}(ss, salt, \text{iteration\_count}, key\_length).$$

In the following section, we investigate the characteristics of the method for generation of the secret key by using the signature of the embedded system.
The entropy of these sets of signatures is presented in Tables 1-4 (the number of component signatures, the length of the system signature and the entropy).

Table 1. The entropy of signatures, sign1 function

<table>
<thead>
<tr>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
</tr>
<tr>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
</tr>
<tr>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
</tr>
<tr>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
</tr>
<tr>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
</tr>
</tbody>
</table>

Table 2. The entropy of signatures, sign2 function

<table>
<thead>
<tr>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
</tr>
<tr>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
</tr>
<tr>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
</tr>
<tr>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
</tr>
<tr>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
</tr>
</tbody>
</table>

For evaluation of the entropy dependence on the used function and on the number of component signatures, we calculate the estimates of entropy: average, standard deviation and prediction interval [22]. The summarized results of the entropy of the signature dependence on the used function are presented in Table 5. As one can see in Table 5, the best quality of signature of the embedded system is obtained by using the function sign4: the maximal value of entropy is 1.000, the least standard deviation is 0.078 and the lowest limit of prediction interval is 0.861. The quality of signatures generated by function sign3 is also relatively high: standard deviation is 0.117 and the lowest limit of the prediction interval is

Table 4. The entropy of signatures, sign4 function

<table>
<thead>
<tr>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
</tr>
<tr>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
</tr>
<tr>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
</tr>
<tr>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
</tr>
<tr>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
</tr>
</tbody>
</table>

Table 3. The entropy of signatures, sign3 function

<table>
<thead>
<tr>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
<th>Bits N</th>
<th>Entro N</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
<td>5</td>
<td>0.954</td>
</tr>
<tr>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
<td>4</td>
<td>0.954</td>
</tr>
<tr>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
<td>3</td>
<td>0.954</td>
</tr>
<tr>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
<td>2</td>
<td>0.954</td>
</tr>
<tr>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
<td>0.954</td>
</tr>
</tbody>
</table>

A. Venčkauskas, N. Usas, J. Mikuckienė, S. Maciulevičius
0.781. Thus, the function of signature generation has to be constructed by using operations XOR and OR.

Table 5. Dependence of the signature entropy on the function

<table>
<thead>
<tr>
<th>Function</th>
<th>Average</th>
<th>Standard deviation</th>
<th>Prediction interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign1</td>
<td>0.894</td>
<td>0.150</td>
<td>0.744</td>
</tr>
<tr>
<td>sign2</td>
<td>0.852</td>
<td>0.115</td>
<td>0.737</td>
</tr>
<tr>
<td>sign3</td>
<td>0.898</td>
<td>0.117</td>
<td>0.781</td>
</tr>
<tr>
<td>sign4</td>
<td>0.939</td>
<td>0.078</td>
<td>0.861</td>
</tr>
</tbody>
</table>

The secret encryption keys are generated from the embedded system signature by using Key Derivation Functions. These functions use hash functions, such as MD5, SHA and SHA-2 etc. Further, we investigate the influence of the hash function algorithm on the value of entropy. Since the signature of the embedded system generated by using sign4 function has the best entropy, we investigate the key generated by this function. The entropy of keys formatted from 2 to 7 component signatures by using sign4 function and MD5, SHA and SHA-2 hash functions is presented in Table 6.

Table 6. The entropy of keys generated by using sign4 function and MD5, SHA and SHA-2 hash functions

<table>
<thead>
<tr>
<th>Test</th>
<th>n=2</th>
<th>n=3</th>
<th>n=4</th>
<th>n=5</th>
<th>n=6</th>
<th>n=7</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.</td>
<td>MD5</td>
<td>SHA</td>
<td>SHA2</td>
<td>MD5</td>
<td>SHA</td>
<td>SHA2</td>
</tr>
<tr>
<td>1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.978</td>
<td>0.997</td>
<td>0.997</td>
</tr>
<tr>
<td>2</td>
<td>0.991</td>
<td>0.999</td>
<td>0.995</td>
<td>0.993</td>
<td>0.996</td>
<td>0.999</td>
</tr>
<tr>
<td>3</td>
<td>0.993</td>
<td>1.000</td>
<td>0.975</td>
<td>0.990</td>
<td>0.997</td>
<td>0.999</td>
</tr>
<tr>
<td>4</td>
<td>0.989</td>
<td>0.993</td>
<td>0.998</td>
<td>0.987</td>
<td>0.996</td>
<td>0.993</td>
</tr>
<tr>
<td>5</td>
<td>0.999</td>
<td>0.999</td>
<td>0.997</td>
<td>0.998</td>
<td>0.996</td>
<td>0.990</td>
</tr>
<tr>
<td>6</td>
<td>1.000</td>
<td>0.996</td>
<td>1.000</td>
<td>0.995</td>
<td>0.998</td>
<td>0.991</td>
</tr>
<tr>
<td>7</td>
<td>0.998</td>
<td>0.998</td>
<td>0.999</td>
<td>0.998</td>
<td>0.996</td>
<td>0.996</td>
</tr>
<tr>
<td>8</td>
<td>0.998</td>
<td>0.996</td>
<td>1.000</td>
<td>0.997</td>
<td>0.999</td>
<td>0.999</td>
</tr>
<tr>
<td>9</td>
<td>0.998</td>
<td>0.991</td>
<td>0.998</td>
<td>0.998</td>
<td>0.993</td>
<td>0.997</td>
</tr>
<tr>
<td>10</td>
<td>0.999</td>
<td>0.999</td>
<td>0.998</td>
<td>0.998</td>
<td>0.991</td>
<td>0.997</td>
</tr>
<tr>
<td>11</td>
<td>1.000</td>
<td>0.994</td>
<td>0.999</td>
<td>0.996</td>
<td>0.998</td>
<td>0.999</td>
</tr>
<tr>
<td>12</td>
<td>0.999</td>
<td>0.999</td>
<td>0.997</td>
<td>0.996</td>
<td>0.998</td>
<td>0.999</td>
</tr>
<tr>
<td>13</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.997</td>
<td>0.995</td>
<td>0.998</td>
</tr>
<tr>
<td>14</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.998</td>
<td>0.994</td>
<td>0.996</td>
</tr>
<tr>
<td>15</td>
<td>0.978</td>
<td>1.000</td>
<td>1.000</td>
<td>0.998</td>
<td>0.993</td>
<td>0.996</td>
</tr>
<tr>
<td>16</td>
<td>0.999</td>
<td>0.999</td>
<td>0.997</td>
<td>0.994</td>
<td>0.995</td>
<td>0.999</td>
</tr>
<tr>
<td>17</td>
<td>0.998</td>
<td>0.998</td>
<td>0.999</td>
<td>0.995</td>
<td>0.996</td>
<td>0.999</td>
</tr>
<tr>
<td>18</td>
<td>0.997</td>
<td>0.991</td>
<td>1.000</td>
<td>0.999</td>
<td>0.998</td>
<td>0.997</td>
</tr>
<tr>
<td>19</td>
<td>0.978</td>
<td>1.000</td>
<td>1.000</td>
<td>0.997</td>
<td>0.993</td>
<td>0.999</td>
</tr>
<tr>
<td>20</td>
<td>1.000</td>
<td>0.998</td>
<td>0.998</td>
<td>1.000</td>
<td>0.996</td>
<td>0.999</td>
</tr>
</tbody>
</table>

Entropy estimates – average, standard deviation and prediction interval, dependence on the used hash function are presented in Table 7.

Table 7. Dependence of the entropy of keys on the used hash function

<table>
<thead>
<tr>
<th>Function</th>
<th>Average</th>
<th>Standard deviation</th>
<th>Prediction interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>0.994</td>
<td>0.008</td>
<td>0.985</td>
</tr>
<tr>
<td>SHA</td>
<td>0.995</td>
<td>0.007</td>
<td>0.988</td>
</tr>
<tr>
<td>SHA-2</td>
<td>0.998</td>
<td>0.003</td>
<td>0.994</td>
</tr>
</tbody>
</table>

All hash functions generate high-entropy cryptographic keys. However, the keys, generated by using function SHA-2, have the least standard deviation (0.003) and the highest of the lowest limit of the prediction interval (0.994).

To summarize, it can be stated that the entropy of the secret encryption key generated by using the signature creation functions based on OR and XOR operations and SHA-2 hash function is highest.

4. Conclusions

In this paper, we present the method for generation of the secret encryption key by using signature of the embedded system and evaluate the entropy of keys and the efficiency of hash functions. The proposed method effectively generates high-entropy keys (entropy value close to 1) without any additional hardware and infrastructure cost, which is vital for the embedded systems with limited resources. The entropy of the secret encryption key generated by using the signature creation functions based on OR and XOR operations and SHA-2 hash function is highest.

In future, we are going to develop a prototype tool of the software protection of the embedded system using the proposed method for generation of the secret
encryption key and investigate the possibilities of its use.

References


Received February 2012.