Entropy Estimation Methods for SW Environments in KCMVP

NSR: Seogchung Seo, Sangwoon Jang
Kookmin University: Yewon Kim, Yongjin Yeom
Contents

- Brief Introduction to KCMVP

- Entropy Estimation Methods for SW Environments in KCMVP
  - Entropy Analysis Framework in KCMVP
  - Correlation-based Entropy Analysis
  - Experimental Results

- Q&A
Brief Introduction to KCMVP (1/3)

KCMVP (Korea Cryptographic Module Validation Program)

- Since 2005
- Validating security and implementation conformance of a CM for protecting sensitive information in public/governmental institutes
- CAVP is conducted in KCMVP process

Standard

- Begin to apply 2015 version since June 2016
- Approved Algorithms: ISO/IEC, KS, TTA
## Approved Algorithms in KCMVP

<table>
<thead>
<tr>
<th>Types</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block cipher</td>
<td>ARIA-128/192/256, LEA-128/192/256, SEED, HIGHT</td>
</tr>
<tr>
<td>Hash function</td>
<td>SHA-224/256/384/512</td>
</tr>
<tr>
<td>MAC</td>
<td></td>
</tr>
<tr>
<td>Hash-based Block cipher-based</td>
<td>HMAC(SHA-224/256/384/512)</td>
</tr>
<tr>
<td>Block cipher-based</td>
<td>GCM(GMAC)</td>
</tr>
<tr>
<td></td>
<td>CCM, CMAC</td>
</tr>
<tr>
<td>RBG</td>
<td></td>
</tr>
<tr>
<td>Hash_DRBG</td>
<td>SHA-224/256/384/512</td>
</tr>
<tr>
<td>HMAC_DRBG</td>
<td>HMAC(SHA-224/256/384/512)</td>
</tr>
<tr>
<td>CTR_DRBG</td>
<td>ARIA-128/192/256, LEA-128/192/256, SEED, HIGHT</td>
</tr>
<tr>
<td>Key establishment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DH: (2048, 224), (2048, 256)</td>
</tr>
<tr>
<td></td>
<td>ECDH: P-224/256, B-233/283, K-233/283</td>
</tr>
<tr>
<td>Public key encryption</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RSAES:</td>
</tr>
<tr>
<td>Digital signature</td>
<td>RSA-PSS, KCDSA, ECDSA, EC-KCDSA</td>
</tr>
</tbody>
</table>
Brief Introduction to KCMVP (3/3)

- **Current Validation Statistics in KCMVP**
  - More than 160 modules have been validated
    - SW modules are dominant in the validation list
    - Almost security level 1

- **Preferred Environments**
  - Windows > Linux/Unix > Java > Mobile (Android, iOS)

- **Most Frequently used approved algorithms in KCMVP**
  - Symmetric-key > Hash > MAC > RBG > Public-key encryption > Digital Signature > KE
Why is Entropy Analysis important?

- Modern Cryptosystems Depends on the Security of Underlying Key
- Security of Key Depends on the Seed Value in DRBG

Not enough entropy in collected noise sources → weakness in whole security systems
Entropy Estimation in S/W Environments

- Characteristics of Noise Source in SW Environments
  - Biased entropy distribution
    - Difficult to expect uniform distribution or IID
  - Dependence among bytes in a source
    - Same data can be repeated
  - Sample size btw several bytes and several hundreds bytes
  - Depends on how to collect and types of OSs
    - Collection interval affects the entropy
  - Generation rate is not consistent
    - Ex) mouse events, key board events and so on

Current methods are suitable for HW noise sources. Thus, entropy evaluation method for SW noise sources is necessary!
Entropy Analysis Framework

- Overall Structure of Entropy Analysis Framework in KCMVP

- Determining entropy noise
- Collecting noise samples
- Correlation-based entropy analysis
- Statistical entropy analysis

Fast feedback to vendors
Providing consistent noise collection means
Providing enhanced accuracy

Under progress of TTA standardization ("Entropy Evaluation Algorithms for Noise Sources in Software Environments")
Correlation-based Entropy Analysis

- Aiming at
  - fast, but moderately accurate entropy analysis for rapid feedback
  - Reducing efforts for gathering noise samples
    - At least, 256 samples required for a noise source

Format of noise sample

```
# of samples  # of byte columns
1 eda2dca75e3245244ef08e31fa31a9eda9cbb7f5c4ac2ce7e6f63c6f90cb5e
2 b23c6cb08b4a07217bc3cf05d006acc986c180fd50078f4c2af0ca7cd1715b9
3 26350b4691f44291323e2b27d4707b1d46680535a14c174f0eda7f49c5d1683
4 bbe37eb96f2fc2759cb934f9ce5719f43418ef92e7b30bd836137028bee550a7
5 397a1abffcfb4b1a2e5be133b60703a357d62ab0c5d5c8e93518d5d46c5ff
6 ad27c34a87ced1f6fc8e278231b2f095c7adff8db74bb3af0e951369a7efa
7 02321c33da0d358a949949098683f9f5b2ae078b683a19c48e6f5b807b772f6f
8 5f7c423ff526b5b484a46bada62d9cd5becc3345373571d45a79e92be2160
9 313270138b5eb77b739214fa871a745e20bd1353312bf3653b30aa27f89e590
...```

```
# of samples  # of byte columns
250 cced1596ecf22a001d2b06faece71a0b7d7238c0c82176bc938ca8e3389aa05
251 60ae59cb79d045e11174d3c296e13231e23089d38e86b6a65aa79038b360f2
252 20717db624d4f2e72a4eb18ef5f7685d6590591ab6f1e021aab64aa0a6a527aa9
253 a9e2b791ed9d078f30ade9c60007b88367c23d35fb135eece2b9245449e305e
254 719832104feff2bcca9e3436caef7c617c88a8dc807b2f3e54d9f230741961812b
255 a6c8a6067fbd721dc6d6b6ccc9897864de3f32ae5d10a161786fbc21e29c4e4d2d
256 3d430457d06048b3ecb7187957c53b1ad5ae981ef66e286561e8dea6819a549```

Correlation-based Entropy Analysis

- **Process of CEA**
  - Distribution-based filtering
    - Byte sample distribution, 0/1’s distribution
  - Byte-oriented entropy analysis
    - Maximum entropy is 8-bit on a byte column
    - Applying Shannon, Min entropy
    - Can be extended to use other statistical entropy analysis
  - Pearson correlation-based entropy reassessment

- **Entropy Computation among bytes**
  - Byte value distribution
  - Shannon entropy
  - Min entropy
  - Computing Pearson correlation among byte columns

- **Entropy reassessment**

---

ICMC 2017
Correlation-based Entropy Analysis

- Distribution-based Filtering
  - Byte values: 8, 16, 32, 64, 128
  - 0/1's values:

- Byte-oriented Entropy Analysis
  - Individual entropy computation
  - Computing Shannon entropy
    - \( \sum_{i=0}^{255} (-P_i \log_2 P_i) \)
  - Computing Min entropy
    - \(-\log_2 (\max(p_0, \ldots, p_{255}))\)
  - Can be extended to other entropy computation methods

<table>
<thead>
<tr>
<th>Sample #</th>
<th>0-th byte</th>
<th>1-th byte</th>
<th>2-th byte</th>
<th>3-th byte</th>
<th>4-th byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0F</td>
<td>01</td>
<td>32</td>
<td>6C</td>
<td>B7</td>
</tr>
<tr>
<td>1</td>
<td>F2</td>
<td>02</td>
<td>F4</td>
<td>7A</td>
<td>F2</td>
</tr>
<tr>
<td>2</td>
<td>CC</td>
<td>01</td>
<td>AB</td>
<td>4F</td>
<td>39</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>D3</td>
<td>73</td>
</tr>
<tr>
<td>255</td>
<td>EF</td>
<td>02</td>
<td>97</td>
<td>46</td>
<td>96</td>
</tr>
<tr>
<td>#Dist</td>
<td>128</td>
<td>5</td>
<td>150</td>
<td>154</td>
<td>180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample#</th>
<th>0-th byte</th>
<th>2-th byte</th>
<th>3-th byte</th>
<th>4-th byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0F</td>
<td>32</td>
<td>6C</td>
<td>B7</td>
</tr>
<tr>
<td>1</td>
<td>F2</td>
<td>F4</td>
<td>7A</td>
<td>F2</td>
</tr>
<tr>
<td>2</td>
<td>CC</td>
<td>AB</td>
<td>4F</td>
<td>39</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>D3</td>
<td>73</td>
</tr>
<tr>
<td>255</td>
<td>EF</td>
<td>97</td>
<td>46</td>
<td>96</td>
</tr>
<tr>
<td>#Dist</td>
<td>128</td>
<td>150</td>
<td>154</td>
<td>180</td>
</tr>
</tbody>
</table>

| Entropy  | e_0       | e_1       | e_2       | e_3       |
Correlation-based Entropy Analysis

- Computing Correlation among Byte Columns

  - \(|\text{cor}_{a,b}| \leq 1\), correlation between a-th and b-th byte columns, \(\text{cor}_{a,a} = 0\)

  \[
  \text{cor}_{a,b} = \frac{\sum_{i=0}^{255} (B_{a,i} - \bar{B}_{a})(B_{b,i} - \bar{B}_{b})}{\sqrt{\sum_{i=0}^{255} (B_{a,i} - \bar{B}_{a})^2 \times \sum_{i=0}^{255} (B_{b,i} - \bar{B}_{b})^2}}
  \]

  - \(k \times k\) correlation table (in this case, 4 \(\times\) 4)
Correlation-based Entropy Analysis

- Entropy Reassessment
  - Basic Idea
    - Total entropy needs to be bigger than individual byte column’s entropy
    \[
    \left( \sum_{i=0}^{k} e_i \right) \geq e_j, \text{ where } 0 \leq j \leq k
    \]
  - General equation

  **Reassessed entropy = original entropy x Reducing factor**

  \[
  e_i' = e_i \left[ 1 - \frac{1}{k} \left( \sum_{j=0}^{k} c_{i,j} \right) \right]
  \]
  where \( i \neq j \) and \( k \) is # of byte columns

  \[
  e'_1 = e_1 \left[ 1 - \frac{1}{4}(|\text{cor}_{12}|+|\text{cor}_{13}|+|\text{cor}_{14}|) \right]
  \]

  \[
  e'_2 = e_2 \left[ 1 - \frac{1}{4}(|\text{cor}_{21}|+|\text{cor}_{23}|+|\text{cor}_{24}|) \right]
  \]

  \[
  e'_3 = e_3 \left[ 1 - \frac{1}{4}(|\text{cor}_{31}|+|\text{cor}_{32}|+|\text{cor}_{34}|) \right]
  \]

  \[
  e'_4 = e_4 \left[ 1 - \frac{1}{4}(|\text{cor}_{41}|+|\text{cor}_{42}|+|\text{cor}_{43}|) \right]
  \]

- Entropy reassessment example (\( k=4 \))
Correlation-based Entropy Analysis

- Developed SW – Entropy Estimator V0.1
Correlation-based Entropy Analysis

- Experimental Result
  - Linux Kernel 2.6 64-bit

<table>
<thead>
<tr>
<th>Without entropy reassessment</th>
<th>Without entropy assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-th byte column</td>
<td>32.000 32.000 32.000 5.797</td>
</tr>
<tr>
<td>1-th byte column</td>
<td>32.000 32.000 32.000 5.676</td>
</tr>
<tr>
<td>15-th byte column</td>
<td>32.000 30.464 17.025 1.892</td>
</tr>
<tr>
<td></td>
<td>22.366 20.643 15.320 3.630</td>
</tr>
</tbody>
</table>

Correlation-based entropy reassessment prevents entropy overestimation!
Standardization of Entropy Evaluation Algorithms for Noise Sources in Software Environments and its Application

2017.05.17.(Wed.)
Kookmin University
Contents

 Abstract

 Standardization : software noise source evaluation method

 Evaluation method of software noise source
Abstract
Abstract

What is RBG (Random Bit Generator)?
- RGB generates a random bit for the cryptographic systems.
- Ideally, the generated random bit is expected to be the result of 'coin toss'.
- It is an essential element in the operation of the cryptographic systems and the cryptographic modules.
- Required properties: unpredictability, non-bias/uniformity, bit-to-bit independence

Vulnerability of Entropy Collection in RBG
- If RBG do not collect enough entropy, the output of a random number is predictable.

Entropy source → Entropy Collection → Entropy Accumulation/Seed Generation → Random Number Generation Algorithm → Output of a Random Sequence

- Collect entropy provided by a noise source.
- Manage the collected entropy.
- Generate a seed.
- Use a standardized cryptographic algorithm.
Abstract

- **Primary Documents and Standards for Entropy Evaluation Methods**
  - **BSI AIS.31**
    - Statistical tests for the entropy sources
    - a basis for validation by CC (Common Criteria for Information Technology Security Evaluation)
  - **NIST SP 800-90B (2nd Draft)**
    - Design and testing requirements for the entropy sources
    - a basis for validation by CMVP (Cryptographic Module Validation Program)
  - **ISO/IEC 20543 (1st CD)**
    - International standard evaluation methodology for entropy

[Relationship between ISO/IEC 20543 and the documents on the evaluation of RNG]

**Hardware Noise Source-Based Entropy Evaluation Method!**
Abstract

- In Korea, **software cryptographic modules** are mainly developed.
  - Software noise sources are collected to generate random numbers.

- **Characteristics of software noise source**
  - The S/W noise source depends on the operating system.
  - It is difficult to expect that the S/W noise source has the uniform distribution or it is IID.
  - (IID : Independent and Identically Distributed)
  - The sample size of the S/W noise source is several bytes to several hundred bytes, but the collected entropy is low.
  - It is difficult to collect the S/W noise source of data size required by primary entropy evaluation methods.
  - The characteristic of a S/W noise source can be greatly changed according to the collection interval of the S/W noise source.

It is not desirable to apply the entropy evaluation method that is suitable for the hardware noise source directly to the software noise source.

**An entropy evaluation method suitable for software noise sources is needed!**
Standardization

: software noise source evaluation method
Standardization: software noise source evaluation method

- It is Korean standard and is registered in TTA (Telecommunications Technology Association).

- Title
  : Entropy Evaluation Algorithms for Noise Sources in Software Environments

- Purpose
  : This standard specifies evaluation and statistical test algorithms for DRBG in software environments.

- Summary
  : This standard specifies resource collecting methods, statistical test algorithms and entropy estimate algorithms in software environments.

- TTA Homepage(eng) : http://www.tta.or.kr/eng/index.jsp
Contents of this TTA standard

1. Statistical test algorithm
   - Statistical test algorithm
   - Health test algorithm

2. Entropy estimate algorithm based on probability theory and information theory
   - Min-entropy estimate algorithm
   - Shannon-entropy estimate algorithm

3. Entropy estimate algorithm based on byte correlation

4. Test vector
1. Statistical Test Algorithm

- **Statistical test algorithm**
  - T1 ~ T5 of Class P1 in AIS.31 were selected for statistical test using small data.
  - Statistical test algorithms are the generalized algorithms that applied P-value to T1 ~ T5.
  - So variable evaluation criterion can be applied these statistical test algorithms.

- **Health test algorithm**
  - Health test algorithms are the algorithms that generalized Repetition Count Test and Adaptive Proportion Test in SP 800-90B.
TTA standard: Entropy Evaluation Algorithms for Noise Sources in S/W Environments

2. Entropy estimate algorithm based on probability theory and information theory
   - **Min-entropy estimate algorithm**
     - The most common value estimate, the collision estimate, the Markov estimate and the compression estimate of entropy estimation for Non-IID data in SP 900-90B were selected, since the software noise source will be Non-IID.
     - Min-entropy estimate algorithms are the algorithms that generalized the selected estimate tests.
   
   - **Shannon-entropy estimate algorithm**
     - There are two types of Shannon-entropy estimate algorithms.
     - One is the generalized T8 of Class P2 in AIS.31.
     - The other is the Shannon-entropy estimate test based on mutual information.
Evaluation method of software noise source
### Evaluation method of software noise source

- **Subject:** GetSystemTime, GetTickCount

<table>
<thead>
<tr>
<th>Windows 7(32bits)</th>
<th>Noise Source</th>
<th>Sample Size</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time-related</strong></td>
<td>GetSystemTime</td>
<td>16</td>
<td>GetSystemTime();</td>
</tr>
<tr>
<td>Noise source</td>
<td>GetTickCount</td>
<td>4</td>
<td>GetTickCount();</td>
</tr>
<tr>
<td></td>
<td>QueryPerformanceCounter</td>
<td>8</td>
<td>QueryPerformanceCounter();</td>
</tr>
<tr>
<td><strong>User-related</strong></td>
<td>GetCursorPos</td>
<td>8</td>
<td>GetCursorPos();</td>
</tr>
<tr>
<td>Noise source</td>
<td>GetCurrentThreadId</td>
<td>4</td>
<td>GetCurrentThreadId();</td>
</tr>
<tr>
<td></td>
<td>GetForegroundWindow</td>
<td>4</td>
<td>GetForegroundWindow();</td>
</tr>
<tr>
<td></td>
<td>GetIcmpStatistics</td>
<td>104</td>
<td>GetIcmpStatistics();</td>
</tr>
<tr>
<td></td>
<td>GetIpStatistics</td>
<td>92</td>
<td>GetIpStatistics();</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceInfo</td>
<td>56</td>
<td>GetPerformanceInfo</td>
</tr>
<tr>
<td></td>
<td>GetProcessHeap</td>
<td>4</td>
<td>GetProcessHeap();</td>
</tr>
<tr>
<td></td>
<td>GetTcpStatistics</td>
<td>60</td>
<td>GetTcpStatistics();</td>
</tr>
<tr>
<td></td>
<td>GetUdpStatistics</td>
<td>20</td>
<td>GetUdpStatistics();</td>
</tr>
<tr>
<td></td>
<td>GlobalMemoryStatusEx</td>
<td>64</td>
<td>GlobalMemoryStatusEx();</td>
</tr>
<tr>
<td></td>
<td>HeapList</td>
<td>32</td>
<td>Heap32ListFirst(); Heap32ListNext();</td>
</tr>
<tr>
<td></td>
<td>ProcessList</td>
<td>1728</td>
<td>Process32First(); Process32ListNext();</td>
</tr>
<tr>
<td></td>
<td>ThreadList</td>
<td>10248</td>
<td>Thread32First(); Thread32ListNext();</td>
</tr>
</tbody>
</table>
Evaluation method of software noise source

- Subject: GetSystemTime, GetTickCount
- Time-related noise sources in Windows OS

Evaluation Scenario

1. Theoretical Analysis
   ➔ Propose the evaluation method for each noise source.
2. Experimental Analysis
   ➔ Validate the proposed evaluation method and propose the collection method.
Theoretical Analysis – Heuristic Analysis in CMVP IG

- A heuristic analysis of time-related noise sources in the Cryptographic Module Validation Implementation (CMVP IG)\(^1\) provided by NIST
  - Representation of time = \( hh : mm : ss.zzz \)
    - \( zzz \): The decimal fraction of a second measured up to the third decimal point (ms, milliseconds).
      - The most variable value.
  - The estimated entropy of time-related noise source is dependent on the frequency of the measurement.
    - If measure at different frequency each time,
      - The number of the \( zzz \) values: 1,000 \( \Rightarrow \) approximately 10 bits of entropy
      - But it is difficult to measure the time at different frequencies each time.
    - If measure at a frequency of about 0.5 seconds each time,
      - The number of the values made out of the second and third “z” : 100
      - The first z has some randomness in it as well.
      - The variability of the \( zzz \) values is similar to having 128. \( \Rightarrow \) 7 bits of entropy
      - The CMVP may even accept a claim of 8 bits of entropy in this case if a slightly more sophisticated argument is made to support such a claim.

\(^1\) Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program (Update: 2016.08.01.)
Theoretical Analysis - GetSystemTime, GetTickCount

- Scenarios for Theoretical Analysis
  1. Analyze the structure and characteristics of each noise source.
     - Refer to Microsoft Developer Network (MSDN) provided by Microsoft.
  2. Analyze heuristically the entropy of each noise source.
     - Use *conservatively* the heuristic analysis in CMVP IG.

- Due to the characteristics of software noise source, the characteristic of a software noise source can be greatly changed according to the collection interval.

The analysis of each noise source is performed considering the collection interval as follows.

- Case of collecting at regular intervals
- Case of collecting at random intervals within a given collection interval
Theoretical Analysis - GetSystemTime

1. Analyze the structure and characteristics of each noise source.

   - GetSystemTime is **the current system date and time**.
     - The System time is expressed in Coordinated Universal Time (UTC).
   - Sample Size: **16 bytes**
   - Collecting function: **GetSystemTime()**
   - Components of GetSystemTime

     | Size  | Component                  |
     |-------|----------------------------|
     | 2 bytes | Year (1601~30827)          |
     | 2 bytes | Month (1~12)               |
     | 2 bytes | Day of Week (0~6)          |
     | 2 bytes | Day (1~31)                 |
     | 2 bytes | Hour (0~23)                |
     | 2 bytes | Minute (0~59)              |
     | 2 bytes | Second (0~59)              |
     | **2 bytes** | **Milliseconds (0~999)** |  

   - The Most variable component: Milliseconds (ms)
   - The heuristic analysis is focused on milliseconds (ms).
2. Analyze heuristically the entropy of each noise source.
   - The time is represented as **hh:mm:ss.zzz** that is the same representation as CMVP IG.
     - The heuristic analysis in CMVP IG is **conservatively used** to estimate entropy.

1) **Case of collecting at regular intervals**
   - The number of the zzz values: 1
     - Ex) When collecting at 160 ms intervals, always collect zzz value increased by 160.
   - Estimated entropy: **0 bit of entropy**

2) **Case of collecting at random intervals within a given collection interval**

<table>
<thead>
<tr>
<th>Given collection interval</th>
<th>Estimated entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 9 ms</td>
<td>0 bit of entropy</td>
</tr>
<tr>
<td>10 ~ 99 ms</td>
<td>3 bits of entropy</td>
</tr>
<tr>
<td>100 ms ~</td>
<td>6 bits of entropy</td>
</tr>
</tbody>
</table>

Ex) When the given collection interval is 20 ms, collect at random intervals within 20 ms.
   - The number of the values made out of the third: 10
   - We consider that there is no randomness in the first and second z. (Conservative perspective more than CMVP IG)
   - The variability of the zzz values is similar to having 10. ➔ **3 bits of entropy**
Theoretical Analysis - GetTickCount

1. Analyze the structure and characteristics of each noise source.
   - GetTickCount is the number of milliseconds that have elapsed since the system was started, up to 49.7 days.
   - Sample Size: 4 bytes
   - Collecting function: `GetTickCount()`
   - The resolution of `GetTickCount()` is typically the range of 10 ms to 16 ms.
     - That is, if `GetTickCount()` is called after a time in the range of 10 ms to 16 ms, `GetTickCount` is increased by that time.
   - The Most variable Byte: LSB(1 byte)
     - The heuristic analysis is focused on LSB(1 byte).
Theoretical Analysis - GetTickCount

2. Analyze heuristically the entropy of each noise source.
   - Assume that GetTickCount is increased by 16 after every 16ms. (Fix the resolution)
     - The heuristic analysis in CMVP IG is conservatively used to estimate entropy.

1) Case of collecting at regular intervals
   - The number of the LSB(1 byte) values: 1
   - Estimated entropy: 0 bit of entropy

2) Case of collecting at random intervals within a given collection interval

<table>
<thead>
<tr>
<th>Given collection interval</th>
<th>Estimated entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 31 ms</td>
<td>0 bit of entropy</td>
</tr>
<tr>
<td>32 ~ 63 ms</td>
<td>1 bit of entropy</td>
</tr>
<tr>
<td>64 ~ 127 ms</td>
<td>2 bits of entropy</td>
</tr>
<tr>
<td>128 ~ 255 ms</td>
<td>3 bits of entropy</td>
</tr>
<tr>
<td>256 ms ~</td>
<td>4 bits of entropy</td>
</tr>
</tbody>
</table>

Ex) When the given collection interval is 160 ms, collect at random intervals within 160 ms.
   - The number of the LSB(1 byte) values: 10
   - Estimated entropy: 3 bit of entropy
Theoretical Analysis - GetTickCount

2. Analyze heuristically the entropy of each noise source.
   - Assume that GetTickCount is increased by that time after a random time in the range of 10 ms to 16 ms.
     - The heuristic analysis in CMVP IG is conservatively used to estimate entropy.

1) Case of collecting at regular intervals
   - The number of the LSB(1 byte) values: 7
   - Estimated entropy: 2 bits of entropy

2) Case of collecting at random intervals within a given collection interval
   - We assume that the entropy of GetTickCount, which is collected at random intervals within a given collection interval, will be estimated higher than the estimated entropy in the previous slide.
Experimental Analysis - GetSystemTime, GetTickCount

1. Collecting the noise source.
   - Extract the most variable 1 byte among sample and Collect.
     - Most entropy estimation test can calculate entropy for samples with the size of 8 bits or less.
     - When selecting the most variable 1 byte position, use the Monobit(Frequency) test and Poker-8 test.

   - Collection Options
     1. Collection interval(ms) : 10, 20, 60, 100, 200, 500
     2. Whether collecting at random intervals within a given collection interval : T/F
     3. The number of samples collecting(byte) : 2000, 5000, 10000, 20000, 50000

   - Select and extract the most variable 1 byte every time it is collected according to each collection option.
Experimental Analysis - GetSystemTime, GetTickCount

2. Entropy estimation method

- Use three algorithms of Min-entropy estimate algorithm of TTA standard.
  - The Most Common Value Estimate Algorithm
  - The Collision Estimate Algorithm
  - The Compression Estimate Algorithm

- The minimum of Min-entropy estimated by each of three algorithms
  \[= \text{Min-entropy of the noise source}\]

- Since the collision estimate algorithm and the compression estimate algorithms use the numerical method, these can operate abnormally.

- So Min-entropy of the noise source is obtained except for the result of the abnormal operation of the algorithm.
  - The result of the abnormal algorithm is represented by this pattern \(\text{\#}\) in the graph.
Experimental Analysis - GetSystemTime

1. **Collect at random interval within given collection interval** and estimate entropy of the collected noise source.

   - Pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

   - 100 ms & 10,000 bytes (①): 0 bit of entropy
     - The most variable 1 byte position selected: **Date-related position**
     - The most variable 1 byte position selected from other results except for: **Milliseconds-related position** (LSB 1 byte)

GetSystemTime is appropriate to extract and collect the 1 byte related to milliseconds in the sample, especially 1 byte located in the LSB(1 byte) position of milliseconds.
Experimental Analysis - GetSystemTime

1. Collect at random interval within given collection interval and Estimate entropy of the collected noise source.

- pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

- Experimental results do not follow the heuristic analysis results.
  - Heuristic Analysis
    - Focus on the number of the zzz values. (zzz = milliseconds)
    - Full-entropy: 10 bits of entropy
  - Experiment
    - Focus on the most variable 1 byte.
    - Full-entropy: 8 bits of entropy

Do heuristic analysis of GetSystemTime with full-entropy of 8 bits.
Experimental Analysis - GetSystemTime

1. Collect at random interval within given collection interval and Estimate entropy of the collected noise source.

- pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

### Heuristic Analysis result

<table>
<thead>
<tr>
<th>Given collection interval</th>
<th>Estimated entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 15 ms</td>
<td>0 bit of entropy</td>
</tr>
<tr>
<td>16 ms ~</td>
<td>4 bits of entropy</td>
</tr>
</tbody>
</table>

- Heuristic analysis of GetSystemTime with full-entropy of 8 bits
  - Focus on LSB(1 byte) of zzz. (zzz = milliseconds)
1. Collect at random interval within given collection interval and estimate entropy of the collected noise source.

- Pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

Experimental results almost follow heuristic analysis results.

Min-entropy of the collected GetSystemTime

\[ \text{Min-entropy of the collected GetSystemTime} = \min (\text{Min-entropy estimated by the estimation algorithm}, \text{Min-entropy estimated by heuristic analysis}) \]
Experimental Analysis - GetSystemTime

2. Collect at regular intervals and Estimate entropy of the collected noise source.

- Pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

$\text{Min-entropy of the collected GetSystemTime} = \min (\text{Min-entropy estimated by the estimation algorithm, Min-entropy estimated by heuristic analysis})$

$= 0 \text{ bit of entropy}$

It is desirable to collect GetSystemTime at a random interval within a given collection interval rather than a regular collection interval.
Experimental Analysis - GetTickCount

1. Collect at random interval within given collection interval and Estimate entropy of the collected noise source.

- pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

- 10 ms & 20,000 bytes (①) or 100 ms & 50,000 bytes (②): 0 bit of entropy
  - The most variable 1 byte position selected: Non-LSB(1 byte)
  - The most variable 1 byte position selected from other results except for: LSB(1 byte)

GetTickCount is appropriate to extract and collect the LSB(1 byte) of the sample.
Experimental Analysis - GetTickCount

1. Collect at random interval within given collection interval and Estimate entropy of the collected noise source.
   - \textbf{pattern :} Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

<table>
<thead>
<tr>
<th>10ms</th>
<th>20ms</th>
<th>60ms</th>
<th>100ms</th>
<th>200ms</th>
<th>500ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000bytes</td>
<td>5000bytes</td>
<td>10000bytes</td>
<td>20000bytes</td>
<td>50000bytes</td>
<td></td>
</tr>
</tbody>
</table>

   - Experimental results almost follow heuristic analysis results.

\[
\text{Min-entropy of the collected GetTickCount} = \min ( \text{Min-entropy estimated by the estimation algorithm}, \\
\quad \text{Min-entropy estimated by heuristic analysis} )
\]
2. Collect at regular intervals and Estimate entropy of the collected noise source.

- pattern: Min-entropy of the noise source obtained except for the result of the abnormal operation of the algorithm.

- Min-entropy of the collected GetTickCount

  \[ \text{Min-entropy of the collected GetTickCount} = \min (\text{Min-entropy estimated by the estimation algorithm, } \text{Min-entropy estimated by heuristic analysis}) \]

  \[ = 0 \text{ bit of entropy} \]

It is desirable to collect GetTickCount at a random interval within a given collection interval rather than a regular collection interval.
Thus...

- **Heuristic Analysis Results**

<table>
<thead>
<tr>
<th>Case of collecting at random intervals within a given collection interval</th>
<th>GetSystemTime (Full-entropy : 10 bits)</th>
<th>GetSystemTime (Full-entropy : 8 bits)</th>
<th>GetTickCount (Full-entropy : 8 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The most variable 1 byte</td>
<td>milliseconds</td>
<td>LSB(1 byte) of milliseconds</td>
<td>LSB(1 byte)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Given Collection Interval</th>
<th>Estimated Entropy</th>
<th>Given Collection Interval</th>
<th>Estimated Entropy</th>
<th>Given Collection Interval</th>
<th>Estimated Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 9 ms</td>
<td>0 bit of entropy</td>
<td>~ 15 ms</td>
<td>0 bit of entropy</td>
<td>~ 31 ms</td>
<td>0 bit of entropy</td>
</tr>
<tr>
<td>10 ~ 99 ms</td>
<td>3 bits of entropy</td>
<td>16 ms ~</td>
<td>4 bits of entropy</td>
<td>32 ~ 63 ms</td>
<td>1 bit of entropy</td>
</tr>
<tr>
<td>100 ms ~</td>
<td>6 bits of entropy</td>
<td></td>
<td></td>
<td>64 ~ 127 ms</td>
<td>2 bits of entropy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128 ~ 255 ms</td>
<td>3 bits of entropy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>256 ms ~</td>
<td>4 bits of entropy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case of collecting at regular intervals</th>
<th>GetSystemTime (Full-entropy : 10 bits)</th>
<th>GetSystemTime (Full-entropy : 8 bits)</th>
<th>GetTickCount (Full-entropy : 8 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 bit of entropy</td>
<td>0 bit of entropy</td>
<td>0 bit of entropy</td>
</tr>
</tbody>
</table>

- **Comparison of experimental results and heuristic analysis results**
  - It is desirable to select the most variable 1 byte position considering the characteristics of the noise source.
  - It is desirable to collect at random intervals within a given collection interval.
  - In order to accurately evaluate the entropy, the heuristic analysis results and the experimental results should be complementary. Min-entropy of the noise source = min(experimental results, heuristic analysis results)
Reference

- NIST, Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program, August, 2016.


- MSDN, https://msdn.microsoft.com/

- KCMVP, http://www.nis.go.kr/AF/1_7_3_3/list.do
Thank you!