A Second View on SecureString 2.0

Günter Fahrnberger

University of Hagen, Universitätsstraße 1, 58097 Hagen, Germany
guenter.fahrnberger@fernuni-hagen.de

Abstract. Many companies have thought about using external hosting solutions. Cloud computing as such a solution attracts prospective users who want to avoid initial costs and standing expenses with the underlying pay-as-you-use model. The outsourcing of sensitive information implies security risks, like eavesdropping and sabotage, for them as soon as they pass any unconfident area. If an outhouse hosting solution serves as data storage only, then an end-to-end cryptosystem without the necessity of having homomorphic properties comes up with the answer. Moreover, secure computations on the encrypted data need the use of more complex cryptosystems. SecureString 1.0 [3] and SecureString 2.0 [4] were proposed as such complex cryptosystems that focus on computing on encrypted character strings in untrustworthy environments (like clouds). While SecureString 1.0 offered a too inflexible approach, SecureString 2.0 as its improvement was introduced textually at a high level only so far. This paper contributes to foster the understanding of SecureString 2.0 by providing performance analysis for its supported operations plus formal definitions, theorems and proofs.

Keywords: blind computing, character string, character string function, character string operation, cloud, cloud computing, secure computing, string function, string operation.

1 Introduction

The outsourcing of own computing power to external places shifts (the responsibility of) the maintenance efforts of the outsourced facilities to a service provider and has been a valid strategy since the commercial use of computers. It began with central computations in mainframes and was supplemented with the hosting of foreign IT-hard- and software. The will to commercialize less utilized or even idle computing resources and the possibility of virtualized hardware has driven the emergence of the topical cloud computing with its three major service models Infrastructure as a service (IaaS), Platform as a service (PaaS) and Software as a service (Saas). Users of cloud computing save maintenance efforts. Furthermore, they also benefit of low or even no initial costs and standing expenses, because almost all cloud providers offer their resources by means of the pay-as-you-use model. The privacy level of outsourced digital data must be maintained somehow to give malicious forces neither the chance to obtain knowledge of sensitive information nor to endanger its integrity. Numeric or textual data intended just to be stored in the cloud without any reading or writing computations on them can protected easily by employing one of the many available sorely approved as secure end-to-end cryptosystems. For (keyword) searches on encrypted character strings, there exist useful schemes in masses as well. The first big challenge are calculations with ciphered numeric operands and results that require either (fully) homomorphic cryptosystems [5,9] or disguising techniques [1]. The second field of research delves for modifying operations on ciphertext character strings. The use of trusted third parties [10], multiple parties [2,7] or cryptographic hardware [6] has led to appropriate solutions, but obviously all of them depend on substantial hardware efforts. Reasons, like the reduced or lost flexibility of clouds, higher costs or the unwillingness of integration by the cloud providers, make the use of additional hardware unattractive or even impossible.

SecureString 1.0 [3] was the first approach to overcome these extra hardware resources with a pure software solution. The detection of considerable limitations in this model ended up in the improved version 2.0 of SecureString [4] whose performance analysis for its supported operations plus its formal definitions, theorems and proofs are focus of interest in this publication.

Section 2 takes a formal look at the en- and decryption scheme of SecureString 2.0.

Section 3 contains the definitions of the three most important character string functions *querying*, *replacing* and *picking* that are supported by SecureString 2.0.

In Section 4 the probabilistic success of statistical attacks against SecureString 2.0 is proved by varying either the n-gram length or the character string length.

Section 5 deals with the time performance of the three character string functions *querying*, *replacing* and *picking*.

Section 6 summarizes and concludes the current document.

2 En- and Decryption Scheme

The en- and decryption scheme of SecureString 2.0 aims to protect the confidentiality of character strings on their way through a P2P-network or a distributed application comprising not only trustworthy nodes. While ciphertext character strings may occur in all nodes and transmission paths, plaintext character strings must not leave a trustworthy node. Therefore, en- and decryptions must be executed in confident environments only, which leads to the following encryption scheme steps in a trustworthy node.

1. Initially, the following input parameters must be chosen: the polygram length $n \in \mathbb{N}$, the number of concurrent coexisting encryption steps $a \in \mathbb{N}$, a filling character which never occurs in plaintext character strings at all and the underlying cryptosystem.
2. All confident nodes agree on a common secret key together.
3. A client repository with an amount of a encryption steps is created. Each of these a steps contains a set of all possible ciphertext n-grams (character strings of length n). Each encrypted n-gram of such a set is synthesized by

appending the same random character string (salt) to the end of the plaintext n-gram and encrypting the merged string with the chosen underlying cryptosystem and its secret key. Therefore, every set of encryption steps gets attached to its own unique salt respectively.
4. An unused encryption step of the client repository is exclusively chosen for each new plaintext character string v.
5. v is split it into its n-grams. If the string length $|v|$ is not dividable by n without having a remainder, then the last substring is be shorter than n characters and must be padded to length n with the defined filling character.
6. Each plaintext n-gram is sought in the client repository and exchanged for its encrypted pendant.
7. Just the ordered conglomeration of the ciphered n-grams represents the wanted ciphertext alternative as result of this encryption scheme.
8. If the number of unused encryption steps in the client repository has fallen below a critical threshold value, then the functionality of the client repository must be maintained by renewing it with maximum a new encryption steps.
9. Continuation with step 4

The previously described encryption scheme gains reversal with the following decryption scheme steps in a trustworthy node.

1. The decisions for the polygram length n, the filling character and the underlying cryptosystem are taken over from the encryption scheme.
2. All confident nodes have agreed on a common secret key together.
3. A new ciphertext character string is split into its ciphered n-grams.
4. Each gained ciphertext n-gram is decrypted separately with the selected underlying cryptosystem and its secret key.
5. Salts and filling characters in the results of the previous step are clipped to plaintext n-grams. The last portion can be shorter than length n of course.
6. Just the ordered sequence of the plaintext n-grams represents the wanted plaintext character string as result of this decryption scheme.
7. Continuation with step 3

While Figure 1 depicts the cryptosystem as output of the previously described en- and decryption scheme, Definition 2 expresses it formally. Definition 1 declares reoccurring notations (emphasized) and their explanations initially as reference in order to enhance the readability of all incident definitions, theorems and proofs.

Definition 1. *Let $\boldsymbol{\Sigma}$ denote an alphabet, let $\mathbf{D} \subseteq \Sigma^*$ denote a dictionary, let the floor function $\lfloor \mathbf{x} \rfloor = max\{y \in \mathbb{Z} | y \leq x\}$ output the largest integer less than or equal to x, let the ceiling function $\lceil \mathbf{x} \rceil = min\{y \in \mathbb{Z} | y \geq x\}$ output the smallest integer greater than or equal to x, let $\mathbf{m} \in \mathbb{N}$ denote the block size of the underlying cryptosystem, let the polygram length $\mathbf{n} \in \mathbb{N}$ denote the number of characters that are salted and encrypted together as a n-gram in ECB (Electronic CodeBook) mode, let $\mathbf{a} \in \mathbb{N} | 0 < a \leq |\Sigma^{m-n}|$ denote the number of concurrent coexisting encryption steps, let $\mathbf{v} \in \Sigma^*$ denote a plaintext character string, let $|\mathbf{v}| \in \mathbb{N}$ denote the length of a string v, let $\mathbf{w} \in \Sigma^{m * \lceil \frac{|v|}{n} \rceil}$ denote the arisen*

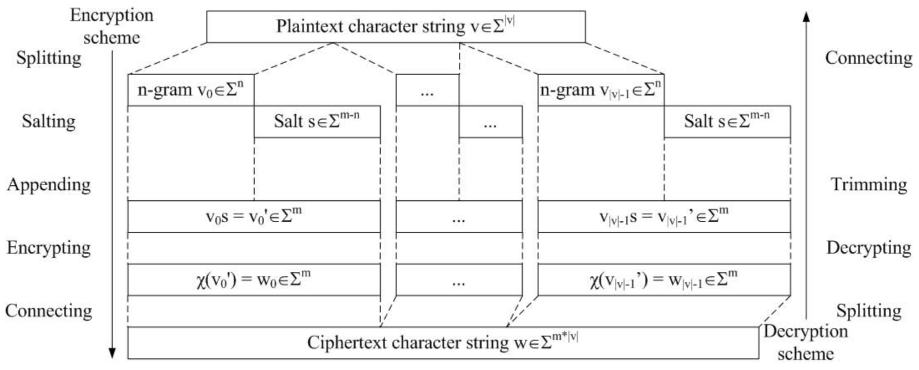

Fig. 1. En- and Decryption Scheme of SecureString 2.0 (n = 1)

ciphertext character string from splitting v into disjunctive n-grams before salting and encrypting them separately during the encryption scheme of SecureString 2.0, opposed to the decryption scheme that takes w, decrypts its ciphered n-grams and removes their salts in order to output v finally, let $\mathbf{C_z} = \frac{1}{z+1}\binom{2z}{z} | z \in \mathbb{N}$ the z^{th} Catalan number, let $\mathbf{u} \in \Sigma^*$ denote a plaintext matching pattern string and $\mathbf{q_0}, \cdots, \mathbf{q_o}, \cdots, \mathbf{q_{a-1}} \in \Sigma^{m*\lceil\frac{|u|}{n}\rceil}$ a set of a appropriate ciphertext alternatives, where $\mathbf{o} \in \mathbb{N}|0 \leq o < a < |\Sigma^{m-n}|$ denotes the o^{th} alternative, let $\mathbf{q_{o_0}}, \cdots, \mathbf{q_{o_p}}, \cdots, \mathbf{q_{o_{\lfloor\frac{|u|}{n}\rfloor}}} \in \Sigma^m$ denote the encrypted n-grams of q_o, where $\mathbf{p} \in \mathbb{N}|0 \leq p < \lceil\frac{|u|}{n}\rceil$ denotes the index of the p^{th} n-gram, let $\mathbf{t} \in \Sigma^*$ denote a plaintext replacement string and $\mathbf{r_0}, \cdots, \mathbf{r_o}, \cdots, \mathbf{r_{a-1}} \in \Sigma^{m*\lceil\frac{|t|}{n}\rceil}$ a set of a appropriate ciphertext alternatives.

Definition 2. *Let Σ, m, n, v, w denote according to Definition 1.*
Then SecureString 2.0 is a polyalphabetic, $|v|$-graphic, $(m*\frac{|v|}{n})$-partite cryptosystem, which encrypts v with a bijective encryption step $\chi : \Sigma^{|v|} \rightarrow \Sigma^{m*\frac{|v|}{n}}$ and decrypts w with a bijective decryption step $\chi^{-1} : \Sigma^{m*\frac{|v|}{n}} \rightarrow \Sigma^{|v|}$ that are based on substitution and straddling.

3 Character String Functions

This section comes along with the formal definitions for the three most important character string functions of SecureString 2.0: *querying*, *replacing* and *picking*.

3.1 Querying

A querying function tests the inclusion of any ciphertext alternative of the specified matching pattern string u in another, advisably longer encrypted string w. A tree or another suitable data structure feeds this operation with the a different ciphertext alternatives for u (see Definition 3). Topical programming languages mostly reserve the method name *contains* for this function.

Definition 3. *Let* $\Sigma, a, j, m, n, o, p, q_0, \cdots, q_o, \cdots, q_{a-1}, q_{o_0}, \cdots, q_{o_p}, \cdots, q_{o_{\lfloor \frac{|u|}{n} \rfloor}}, u, v, w$ *denote according to Definition 1, and let* $w_0, \cdots, w_j, \cdots, w_{\lfloor \frac{|v|}{n} \rfloor} \in \Sigma^m$ *denote the encrypted n-grams of w, where* $j \in \mathbb{N} | 0 \leq j < \lceil \frac{|v|}{n} \rceil$ *denotes the index of the* j^{th} *n-gram, then w is queried by*

$$contains : \Sigma^{m*\lceil \frac{|v|}{n} \rceil} \times \Sigma^{m*\lceil \frac{|u|}{n} \rceil} \to \{false, true\},$$

$$(w, q_0, \cdots, q_{a-1}) \mapsto contains((w, q_0, \cdots, q_{a-1}))$$

$$:= \begin{cases} true & if\ (\exists o \in \mathbb{N})(\forall p \in \{0, \cdots, \lfloor \frac{|u|}{n} \rfloor\})(\exists j \in \{0, \cdots, \lfloor \frac{|v|}{n} \rfloor\}) w_j = q_{o_p} \\ false & otherwise \end{cases}$$

3.2 Replacing

Initially, replacing does the same as querying a ciphered string w. Additionally, it exchanges occurring ciphertext alternatives of the specified matching pattern string u for ciphertext alternatives of the specified replacement string t (see Definition 4). A function, which replaces just the first occurrence of a ciphertext alternative of u with such one of t, is usually named *replaceFirst*. Such one, which replaces all occurrences of ciphertext alternatives of u with such ones of t, is commonly called *replaceAll*. Each replace function demands two trees or equally good data structures as arguments. The first one contains the a encrypted alternatives for u, and the second one possesses those ones for t. The replacement t can be the empty string ϵ as well, which ends in cutting out the first respectively all occurring ciphertext alternative(s) of u.

Definition 4. *Let* $\Sigma, a, j, m, n, o, p, q, q_0, \cdots, q_o, \cdots, q_{a-1}, q_{o_0}, \cdots, q_{o_p}, \cdots, q_{o_{\lfloor \frac{|u|}{n} \rfloor}}, r_0, \cdots, r_o, \cdots, r_{a-1}, u, v, w$ *denote according to Definition 1, let* $w_0, \cdots, w_{j_0}, q_o, w_{j_0 + \lceil \frac{|u|}{n} \rceil + 1}, \cdots, w_{j_y}, q_o, w_{j_y + \lceil \frac{|u|}{n} \rceil + 1}, \cdots, w_{\lfloor \frac{|v|}{n} \rfloor} \in \Sigma^m$ $| y \in \mathbb{N}$ *denote the encrypted n-grams of w that includes* $y+1$ *occurrences of* q_o *obviously, then the first occurrence of* q_o *in w is exchanged for* r_o *by*

$$replaceFirst : \Sigma^{m*\lceil \frac{|v|}{n} \rceil} \times \Sigma^{m*\lceil \frac{|u|}{n} \rceil} \times \Sigma^{m*\lceil \frac{|t|}{n} \rceil} \to \Sigma^{\{m*\lceil \frac{|v|}{n} \rceil, m*\lceil \frac{|v-u+t|}{n} \rceil\}},$$

$$(w, q_0, \cdots, q_{a-1}, r_0, \cdots, r_{a-1}) \mapsto replaceFirst((w, q_0, \cdots, q_{a-1}, r_0, \cdots, r_{a-1}))$$

$$:= \begin{cases} w\ if\ contains(w, q_0, \cdots, q_{a-1}) = false \\ w_0 \cdots w_{j_0} r_o w_{j_0 + \lceil \frac{|u|}{n} \rceil + 1} \cdots w_{j_y} q_o w_{j_y + \lceil \frac{|u|}{n} \rceil + 1} \cdots w_{\lfloor \frac{|v|}{n} \rfloor} \quad otherwise \end{cases}$$

or all occurrences of q_o *in w are exchanged for* r_o *by*

$$replaceAll : \Sigma^{m*\lceil \frac{|v|}{n} \rceil} \times \Sigma^{m*\lceil \frac{|u|}{n} \rceil} \times \Sigma^{m*\lceil \frac{|t|}{n} \rceil} \to \Sigma^*,$$

$$(w, q_0, \cdots, q_{a-1}, r_0, \cdots, r_{a-1}) \mapsto replaceAll((w, q_0, \cdots, q_{a-1}, r_0, \cdots, r_{a-1}))$$

$$:= \begin{cases} w\ if\ contains(w, q_0, \cdots, q_{a-1}) = false \\ w_0 \cdots w_{j_0} r_o w_{j_0 + \lceil \frac{|u|}{n} \rceil + 1} \cdots w_{j_y} r_o w_{j_y + \lceil \frac{|u|}{n} \rceil + 1} \cdots w_{\lfloor \frac{|v|}{n} \rfloor} \quad otherwise \end{cases}$$

3.3 Picking

A picking function, which is known under the name of *substring* generally, outputs a new string that is substring of an encrypted string w (see Definition 5). A substring function needs two arguments: the position of the first and of the last substring character in w. SecureString supports the second known kind of *substring* as well which is called with a start index only in order to use the last character of w as ending index implicitly.

Definition 5. *Let Σ, n, v, w denote according to Definition 1, let $d \in \mathbb{N}$ denote the beginning index, let $e \in \mathbb{N}$ denote the ending index, and let $w_0, \cdots, w_d, \cdots, w_e, \cdots, w_{\lfloor \frac{|v|}{n} \rfloor} \in \Sigma^m$ denote the encrypted n-grams of w, where $0 \le d \le e \le \lfloor \frac{|v|}{n} \rfloor$, then a substring of w is picked out by*

$$substring : \Sigma^{m*\lceil \frac{|v|}{n} \rceil} \times \mathbb{N} \times \mathbb{N} \to \Sigma^*, (w, d, e) \mapsto substring((w, d, e)) :=$$

$$w_d, \cdots, w_{e-1}$$

or by

$$substring : \Sigma^{m*\lceil \frac{|v|}{n} \rceil} \times \mathbb{N} \to \Sigma^*, (w, d) \mapsto substring((w, d)) := w_d, \cdots, w_{\lfloor \frac{|v|}{n} \rfloor}$$

4 Threat Model

The initial publication of SecureString 2.0 [4] demonstrated the resistance of SecureString 2.0 against ciphertext-only-, known-plaintext-, (adaptive) chosen-plaintext-, (adaptive) chosen-ciphertext-attacks and a combination of the both latter ones as long as the secret key keeps concealed. This section addresses the evaluation of the success probability of two ciphertext-only attacks that are directed against cryptosystems based on substitution. Even if an opponent uncovered a single cloud repository or ciphertext character string, e.g. with one of these two ciphertext-only offenses, SecureString 2.0 would heal itself by using salts only once and exchanging its exhausted repositories. Hence the prudent design of SecureString 2.0 achieves forward and backward secrecy because compromised strings or repositories do not expose their predecessors or successors.

4.1 *n*-gram Repetition Pattern Attack

The first attack compares the n-gram repetition pattern of a ciphertext character string with all known n-gram repetition patterns of an appropriate dictionary D and the creation of a set of fitting plaintext character strings. A lack of reference repetition patterns would make this attack unworkable. While n is varying during the n-gram repetition pattern attack in Theorem 1, the character string length is changing in Theorem 2.

n-gram Repetition Pattern Attack with Variation of n

Theorem 1. *Let $\Sigma, D, m, n, o, q_o, r_o, t, u, v, w$ denote according to Definition 1. If n increases linearly, then the number of fitting plaintext character strings in D as result of a n-gram repetition pattern attack against w, q_o or r_o increases polynomially, and therefore the probability to reveal v, u or t decreases polynomially.*

The proof is done representatively for v/w-pairs by mathematical induction.

Basis ($n = 1$): The number of repetition patterns, to which all SecureString 2.0-ciphertext strings of length $m*\lceil\frac{|v|}{n}\rceil = m*\lceil\frac{|v|}{1}\rceil = m*|v|$ in D are distributed, is the $|v|^{th}$ Catalan number $C_{|v|} = \frac{(2*|v|)!}{(|v|+1)!*|v|!}$.

Induction step ($n \to n+1$): The number of repetition patterns, to which all SecureString 2.0-ciphertext strings of length $m * \lceil\frac{|v|}{n+1}\rceil$ in D are distributed, is the $\lceil\frac{|v|}{n+1}\rceil^{th}$ Catalan number $C_{\lceil\frac{|v|}{n+1}\rceil} = \frac{(2*\lceil\frac{|v|}{n+1}\rceil)!}{(\lceil\frac{|v|}{n+1}\rceil+1)!*\lceil\frac{|v|}{n+1}\rceil!}$. $C_{\lceil\frac{|v|}{n+1}\rceil}$ is

$$\frac{C_{\lceil\frac{|v|}{n}\rceil}}{C_{\lceil\frac{|v|}{n+1}\rceil}} = \frac{\frac{(2*\lceil\frac{|v|}{n}\rceil)!}{(\lceil\frac{|v|}{n}\rceil+1)!*\lceil\frac{|v|}{n}\rceil!}}{\frac{(2*\lceil\frac{|v|}{n+1}\rceil)!}{(\lceil\frac{|v|}{n+1}\rceil+1)!*\lceil\frac{|v|}{n+1}\rceil!}} = \frac{(2*\lceil\frac{|v|}{n}\rceil)!*(\lceil\frac{|v|}{n+1}\rceil+1)!*\lceil\frac{|v|}{n+1}\rceil!}{(2*\lceil\frac{|v|}{n+1}\rceil)!*(\lceil\frac{|v|}{n}\rceil+1)!*\lceil\frac{|v|}{n}\rceil!} =$$

$$= \begin{cases} 1 & \text{if } \lceil\frac{|v|}{n+1}\rceil = \lceil\frac{|v|}{n}\rceil \\ \frac{2*(2*\lceil\frac{|v|}{n}\rceil-1)}{\lceil\frac{|v|}{n}\rceil+1} & \text{otherwise} \end{cases}$$

times lower than $C_{\lceil\frac{|v|}{n}\rceil}$.

Accordingly, the incidence probability per repetition pattern increases polynomially if n increases linearly. Therefore, the probability to reveal v decreases polynomially. □

n-gram Repetition Pattern Attack with Variation of Character String Length

Theorem 2. *Let $\Sigma, D, m, n, o, q_o, r_o, t, u, v, w$ denote according to Definition 1. If a length $|w|$, $|q_o|$ or $|r_o|$ increases linearly, then the number of fitting repetition patterns in D as result of a n-gram repetition pattern attack against w, q_o or r_o increases polynomially, and therefore the probability to reveal v, u or t decreases polynomially.*

The proof is done representatively for v/w-pairs by mathematical induction.

Basis ($|v| = 1$): The number of repetition patterns, to which all SecureString 2.0-ciphertext strings of length $m * \lceil\frac{|v|}{n}\rceil = m * \lceil\frac{1}{n}\rceil = m$ in D are distributed, is the $|v|^{th} = 1^{st}$ Catalan number $C_1 = \frac{(2*|v|)!}{(|v|+1)!*|v|!} = \frac{(2*1)!}{(1+1)!*1!} = \frac{2!}{2!} = 1$.

Induction step ($|v| \to |v|+1$): The number of repetition patterns, to which all plaintext character strings of length $m * \lceil\frac{|v|+1}{n}\rceil$ in D are distributed, is the $\lceil\frac{|v|+1}{n}\rceil^{th}$ Catalan number $C_{\lceil\frac{|v|+1}{n}\rceil} = \frac{(2*\lceil\frac{|v|+1}{n}\rceil)!}{(\lceil\frac{|v|+1}{n}\rceil+1)!*\lceil\frac{|v|+1}{n}\rceil!}$.

$C_{\lceil \frac{|v|+1}{n} \rceil}$ is

$$\frac{C_{\lceil \frac{|v|+1}{n} \rceil}}{C_{\lceil \frac{|v|}{n} \rceil}} = \frac{\frac{(2*\lceil \frac{|v|+1}{n} \rceil)!}{(\lceil \frac{|v|+1}{n} \rceil+1)!*\lceil \frac{|v|+1}{n} \rceil!}}{\frac{(2*\lceil \frac{|v|}{n} \rceil)!}{(\lceil \frac{|v|}{n} \rceil+1)!*\lceil \frac{|v|}{n} \rceil!}} = \frac{(2*\lceil \frac{|v|+1}{n} \rceil)! * (\lceil \frac{|v|}{n} \rceil + 1)! * \lceil \frac{|v|}{n} \rceil!}{(2*\lceil \frac{|v|}{n} \rceil)! * (\lceil \frac{|v|+1}{n} \rceil + 1)! * \lceil \frac{|v|+1}{n} \rceil!} =$$

$$= \begin{cases} 1 & \text{if } \lceil \frac{|v|+1}{n} \rceil = \lceil \frac{|v|}{n} \rceil \\ \frac{2*(2*\lceil \frac{|v|+1}{n} \rceil - 1)}{\lceil \frac{|v|+1}{n} \rceil + 1} & \text{otherwise} \end{cases}$$

times higher than $C_{\lceil \frac{|v|}{n} \rceil}$.

Accordingly, the incidence probability per repetition pattern decreases polynomially if $|v|$ increases linearly, and therefore the probability to reveal v increases polynomially. This proof assumes the same number of dictionary words per character string length. In many dictionaries the number of character strings per length resembles almost normal distribution. Such a case opposes a decreasing number of words to an increasing number of repetition patterns for character string lengths larger than the median length. Thus the incidence probability per repetition pattern sinks respectively the probability to reveal v grows even faster than proved. □

4.2 *n*-gram Distribution Attack

The second attack compares the n-gram distribution of a ciphertext character string with the overall n-gram distribution of an appropriate dictionary D statistically, e.g. with Fisher's exact test [8]. This attack becomes infeasible if the required reference n-gram distribution is unavailable. While n is varying during the n-gram distribution attack in Theorem 3, the character string length is changing in Theorem 4.

n-gram Distribution Attack with Variation of *n*

Theorem 3. *Let $\Sigma, D, m, n, o, q_o, r_o, t, u, v, w$ denote according to Definition 1. If n increases linearly, then the probability to reveal v, u or t statistically decreases.*

The proof is done representatively for v/w-pairs by mathematical induction.

Basis ($n = 1$): Each SecureString 2.0-ciphertext string of length $m * \lceil \frac{|v|}{n} \rceil = m * \lceil \frac{|v|}{1} \rceil = m * |v|$ in D consists of $\lceil \frac{|v|}{n} \rceil = \lceil \frac{|v|}{1} \rceil = |v|$ monograms (1-grams).

Induction step ($n \to n+1$): Each SecureString 2.0-ciphertext string of length $m * \lceil \frac{|v|}{n+1} \rceil$ in D consists of $\lceil \frac{|v|}{n+1} \rceil$ $(n+1)$-grams.

$\lceil \frac{|v|}{n+1} \rceil$ is

$$\frac{\lceil \frac{|v|}{n} \rceil}{\lceil \frac{|v|}{n+1} \rceil} = \begin{cases} 1 & \text{if } \lceil \frac{|v|}{n+1} \rceil = \lceil \frac{|v|}{n} \rceil \\ \frac{n+1}{n} = 1 + \frac{1}{n} & \text{otherwise} \end{cases}$$

times lower than $\lceil \frac{|v|}{n} \rceil$.

Accordingly, lowering the number of included grams per ciphertext string reduces the probability to reveal v statistically. $\square$

n-gram Distribution Attack with Variation of Character String Length

Theorem 4. *Let $\Sigma, D, m, n, o, q_o, r_o, t, u, v, w$ denote according to Definition 1. If a length $|w|$, $|q_o|$ or $|r_o|$ increases linearly, then the probability to reveal v, u or t statistically increases.*

The proof is done representatively for v/w-pairs by mathematical induction.

Basis ($|v| = 1$): Each SecureString 2.0-ciphertext string of length $m * \lceil \frac{|v|}{n} \rceil = m * \lceil \frac{1}{n} \rceil = m$ in D consists of $\lceil \frac{|v|}{n} \rceil = \lceil \frac{1}{n} \rceil = 1$ n-gram.

Induction step ($|v| \to |v| + 1$): Each SecureString 2.0-ciphertext string of length $m * \lceil \frac{|v|+1}{n} \rceil$ in D consists of $\lceil \frac{|v|+1}{n} \rceil$ n-grams.

$\lceil \frac{|v|+1}{n} \rceil$ is

$$\frac{\lceil \frac{|v|+1}{n} \rceil}{\lceil \frac{|v|}{n} \rceil} = \begin{cases} 1 & \text{if } \lceil \frac{|v|+1}{n} \rceil = \lceil \frac{|v|}{n} \rceil \\ \frac{|v|+1}{|v|} = 1 + \frac{1}{|v|} & \text{otherwise} \end{cases}$$

times higher than $\lceil \frac{|v|}{n} \rceil$.

Accordingly, a longer character string causes a higher number of grams in average and therefore abets the probability to reveal w statistically. $\square$

5 Performance

This section evaluates the time performance of the three most important character string functions of SecureString 2.0: *querying*, *replacing* and *picking*. Additionally, the investigation confronts the obtained mean turnaround times of SecureString 2.0 with those ones of comparable cryptosystems, among them its predecessor SecureString 1.0 and AES (Advanced Encryption Standard) as bare transport cryptosystem. In contrast to the statistics of SecureString 1.0, those ones of SecureString 2.0 were drawn for $n = 1$ only, because queried, inserted or deleted substrings need not to follow the n-gram bounds if $n > 1$, and therefore they cause too time-consuming inter-n-gram-operations despite better privacy. For example let be $n = 2, u =$'*bc*'$, v =$'*abcd*', then $q = \chi($'*bc*'$)$ must be found in $w = \chi($'*ab*'$)\chi($'*cd*'$)$, even if '*b*' and '*c*' are parts of different n-grams obviously. The memory consumptions of both SecureString-versions behave identically and thus require a citation of the performance analysis for SecureString 1.0 [3] only.

5.1 Querying Performance

Firstly, the querying performance of SecureString 2.0 is tested for character string lengths $1 \leq |v| \leq 64$ and all possible matching pattern string lengths $1 \leq$

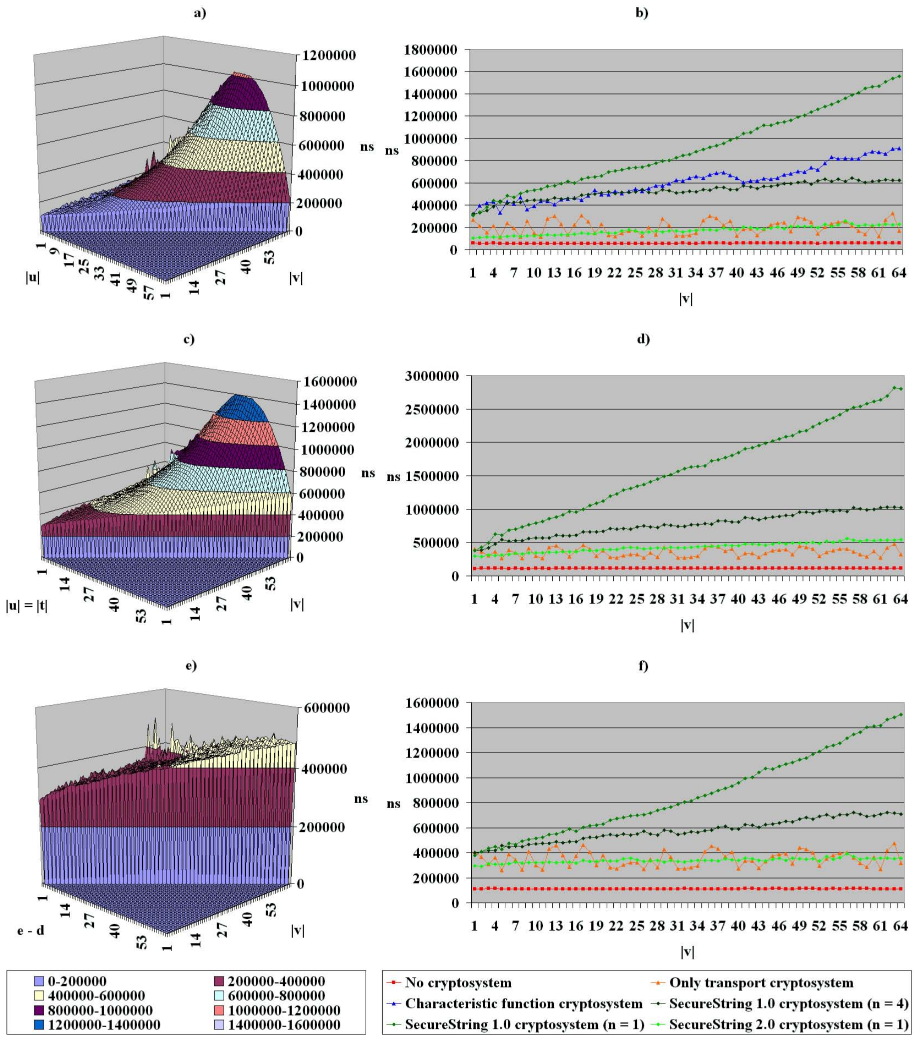

Fig. 2. Performance of Character String Functions

$|u| \leq |v|$. For each data value in Figure 2 a) the querying operation is conducted for 1.000.000 random ciphertext character string/matching pattern string-pair samples and the mean value shown in nanoseconds. As a result, the time complexity for querying functions turns out to be $O(|v| * \log(\min(|u|, |v| - |u|))$.

Secondly, the inclusion of random matching pattern strings of length $|u| = \min(4, |v|)$ is tested in 1.000.000 random character string samples per length $1 \leq |v| \leq 64$. This is evaluated in nanoseconds per operation. It can be easily observed by means of Figure 2 b) that the performance of SecureString 2.0 behaves like that one of a transport cryptosystem and much more better than that one of SecureString 1.0.

5.2 Replacing Performance

Like for querying, the performance of replacing operations with SecureString 2.0 is measured for character string lengths $1 \leq |v| \leq 64$ and all possible matching pattern string lengths $1 \leq |u| \leq |v|$ firstly. Each found matching pattern string is replaced by a replacement string of equal length $|t| = |u|$. Each data value in Figure 2 c) represents the mean value of 1.000.000 random ciphertext character string/matching pattern string/replacement string-triplets in nanoseconds. The time complexity can be recognized with $O(|v| * \log(\min(|u|, |v| - |u|))$ again.

In the second part of the performance analysis for the replacing operation, random matching patterns strings of length $|u| = \min(4, |v|)$ are replaced by random equally long replacement strings in 1.000.000 random character string samples per length $1 \leq |v| \leq 64$. Again, this is measured in nanoseconds per operation. Figure 2 d) displays that similar behavior as for the previous querying operation can be observed: SecureString 2.0 displays lower efforts than SecureString 1.0. Reference values for the characteristic function cryptosystem are omitted here because it lacks to support replacing functions.

5.3 Picking Performance

Initially, the performance of picking operations is examined by picking out substrings of all possible lengths $1 \leq e - d \leq |v|$, where d is a random left and e a random right border, out of character strings of length $1 \leq |v| \leq 64$. Each data mean value in Figure 2 e) consists of 1.000.000 samples and is scaled in nanoseconds. An approximate time complexity of $O(|v|)$ can be anticipated.

The comparison of the different cryptosystems for the picking operation is performed by cutting out substrings of length $\min(4, |v|)$ at random beginning indexes in 1.000.000 random character string samples per length $1 \leq |v| \leq 64$. Yet again, the measurement is taken in nanoseconds per operation. Figure 2 f) displays that SecureString 2.0 also surpasses the picking performance of SecureString 1.0. Of course, both SecureString versions perform worse than a transport cryptosystem, but provide privacy within the cloud application.

6 Conclusion

The cryptosystem SecureString 2.0 offers querying and modifying functions on encrypted character strings in untrustworthy environments (like clouds).

After defining the en- and decryption scheme and the three most important supported character string functions *querying*, *replacing* and *picking* formally, it was shown that the success probability to break an encrypted character string (with perceptible boundaries) by a n-gram distribution attack or a statistical n-gram distribution attack decreases with an increasing number of commonly ciphered characters n respectively increases with an increasing character string length. Inferred from these results, an implementer of SecureString 2.0 ought be firmly encouraged to use one salt only once and only to protect a single character string. Nevertheless, if a text of SecureStrings is conveyed or processed together, then each ciphered delimiter between two words or sentences, e.g. a blank or a full stop, can share the salt with its previously neighbored character string safely without risking detection of its meaning. Conducted analysis for three character string functions exposed improved time performance of SecureString 2.0 compared to SecureString 1.0.

In short, SecureString 2.0 offers a good trade-off between privacy and time performance. Nevertheless, better protection for the integrity of ciphertext character strings can be obtained, e.g. against cut and splice or replay attacks, by assembling a random nonce, a timestamp and an incrementing counter in each seeded salt.

References

1. Atallah, M.J., Pantazopoulos, K., Rice, J.R., Spafford, E.E.: Secure outsourcing of scientific computations. In: Zelkowitz, M.V. (ed.) Trends in Software Engineering. Advances in Computers, vol. 54, pp. 215–272. Elsevier (2002)
2. Brun, Y., Medvidovic, N.: Keeping data private while computing in the cloud. In: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), pp. 285–294 (2012)
3. Fahrnberger, G.: Computing on encrypted character strings in clouds. In: Hota, C., Srimani, P.K. (eds.) ICDCIT 2013. LNCS, vol. 7753, pp. 244–254. Springer, Heidelberg (2013)
4. Fahrnberger, G.: Securestring 2.0 - a cryptosystem for computing on encrypted character strings in clouds. In: Innovative Internet Community Systems, VDI Düsseldorf (2013)
5. Goluch, S.: The development of homomorphic cryptography - from rsa to gentrys privacy homomorphism. Master's thesis, Vienna University of Technology (2011)
6. Itani, W., Kayssi, A., Chehab, A.: Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures. In: Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2009, pp. 711–716 (2009)
7. Maheshwari, N., Kiyawat, K.: Structural framing of protocol for secure multiparty cloud computation. In: 2011 Fifth Asia Modelling Symposium (AMS), pp. 187–192 (2011)
8. Mehta, C.R., Patel, N.R.: A network algorithm for performing fisher's exact test in r x c contingency tables. Journal of the American Statistical Association 78(382), 427–434 (1983)
9. Rodríguez-Silva, D.A., González-Castaño, F.J., Adkinson-Orellana, L., Fernández-Cordeiro, A., Troncoso-Pastoriza, J.R., González-Martínez, D.: Encrypted domain processing for cloud privacy - concept and practical experience. In: CLOSER, pp. 591–596 (2011)
10. Wei, L., Zhu, H., Cao, Z., Jia, W., Vasilakos, A.: Seccloud: Bridging secure storage and computation in cloud. In: 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 52–61 (2010)