Protecting the Privacy with Human-Readable Pseudonyms: One-Way Pseudonym Calculation on Base of Primitive Roots

Uwe Roth
SANTEC
CRP Henri Tudor
Luxembourg, Luxemburg
uwe.roth@tudor.lu

***Abstract*—Pseudonyms are used in medical data to protect the privacy of patients: Demographics like name, gender and age are removed from the medical data and are replaced by a unique pseudonym. Medical data with the same pseudonym belongs to the same person. A pseudonym should be random or at least pseudo-random and should not allow drawing conclusions about the identity of the patient. Random pseudonyms are not always possible and therefore must be somehow calculated out of an identifier of the patient, e.g., by hashing or encrypting the identifier of the patient. In this paper an alternative algorithm is proposed to calculate pseudonyms on the base of primitive roots. The algorithm guarantees a collision free pseudo-random distribution of the pseudonyms, but creates pseudonyms with a bit-depth that easily can be transformed to a human readable representation. This is important if the pseudonym has to be used (be read, be written) in a day-to-day workflow. The pseudonymisation algorithm acts as a one-way function if all of the calculation parameters are kept secret.**

Keywords-patient privacy-enhancing technologies; secure patient data storage; pseudonymisation; one-way function; primitive root.

I. Introduction

Pseudonymisation is a process where demographics and identifier of a person are removed out of an information record and replaced by a pseudonym. This step is demanded to protect the privacy of patients in cases of secondary usage of medical data, e.g., for research or statistical purposes. In these cases knowledge about the identity of the person is unnecessary and therefore must be protected against disclosure. In contrast to anonymisation, a pseudonym allows to link data from several sources to the same person, which helps to improve the quality of the research or statistics.

An example for the need of pseudonymization is the storage of medical data, samples, blood and urine in biobanks. Researchers are not interested in the identity of the person behind this material. But a pseudonym is needed to link the samples from the same person, which have been taken at different locations and during different collection events. The pseudonym will not only allow the linkage to the same person but also allows protecting the identity of the patient behind the sensitive data.

Concepts on how identities of patients and their pseudonyms are used and managed (including identity matching, identity vigilance, linkage of identifiers from different domains) to securely exchange data are discussed in many publications (e.g., [1] and [2]). These concepts are not in the focus of this paper.

Assuming, that all the problems of identity matching are solved, the pseudonym number or pseudonym string itself has to be calculated or determinate at one point. There are several options to create a pseudonym with a given set of demographics. Some of these techniques base on hashing or encryption of a unique identifying number of the person. Others simply chose a random number and link this number with the identity.

Things are getting difficult if the pseudonym must be short enough to be human readable and usable, which limits the number of effective bits of the pseudonym to approx. 32 to 40 bits (equivalent 6-8 chars). Current hashing and encryption algorithms work with 128 bits minimum. In that case, the outcome of the process must be cropped to the desired bit-length, which leads to an unpredictable risk for pseudonym collisions.

Adopted solutions that take the small number of bits into account (e.g., [3]) still base on techniques that are used in symmetric encryption or hashing algorithms like the Advanced Encryption Standard AES [4] or the Secure Hash Algorithm SHA [5] (permutation, rotation, transformation, diffusion). With the small bit-depth of the initial data and the missing of deeper cryptanalysis, it is difficult to estimate how secure these algorithms finally are and how difficult it is to re-compute the person identifier with a given pseudonym.

The solution described in this paper is based on asymmetric encryption techniques as it is used as a one-way trapdoor-function. Because the used technique is hard, the described pseudonymisation algorithm in total is also hard.

The paper is structured as follows:

In Section II – Methods, firstly, a short introduction into the mathematical foundations of primitive roots is given. Then it is shown that this approach can be used for the calculation of pseudonyms. The section ends with suggestions how to find a necessary primitive root and how the calculations can be implemented fast and efficient.

In Section III – Results, the bit depth of the secrets that are used in the pseudonym calculation is shown especially if these secrets are sufficient to protect the original person identifier. A special focus is made on the human readability of the pseudonym and the speed to calculate the pseudonym.

In Section IV – Discussion, potential attacks and re-identification risks are discussed and an example for the applicability is given.

The paper ends with a conclusion in Section V.

II. Methods

The mathematics behind the pseudonym calculation of a person ID is based on primitive roots of prime numbers as it is used in the Diffie-Hellman protocol to ensure a secure key exchange [6].

A. Modulo operation (mod)

a mod b, with *a* and *b* being positive integer numbers is defined as the remainder of the division of the dividend *a* and the devisor *b*. The result of the mod operation is an integer value in the range from *0* to *b-1*.

Example: 112 mod 24 = 16 because 112/24 = 4 with the remainder 16.

B. Discrete logarithm

Having the equation:

$$b = a^i \, mod \, p, \text{ with } p \text{ prime}, \; i \in \{1..p\text{-}1\} \qquad (1)$$

Then *i* is called the discrete logarithm, which is equivalent to

$$i = log_a \, b \qquad (2)$$

The calculation of *b* is easy but currently there exists no efficient way to find the discrete logarithm *i* with given *a*, *b* and *p*.

This statement is only true if *p* is big enough to make the use of pre-calculated solution tables impossible and if no pre-knowledge about *i* exists that allows reducing the search space.

C. Primitive roots

The property of *a* being a primitive root of prime *p* means, that

$$a^i \, mod \, p, \text{ with } i = 1..p\text{-}1 \qquad (3)$$

results in all values of *1..p-1*, with no value double or missing. This property is relevant to create collision free pseudonyms.

Primitive roots have been used already a long time ago to create good random number generators [7]. Our algorithm uses this knowledge to introduce pseudo-randomness into the series of pseudonyms.

D. Adaption for the pseudonymisation calculation

With *k* bits that are reserved for the pseudonym, a prime number *p* should be chosen that in best case is the highest prime number lower than 2^k. With the given *p*, the interval of possible person IDs and pseudonyms is *1..p-1*. The numbers which are invalid in the *k*-bit number space are 0 and $p..2^k\text{-}1$. As an example: For *k=31*, the highest prime lower than 2^{31} is $2^{31}\text{-}1$. In this case, only *0* and $2^{31}\text{-}1$ cannot be used as person IDs or pseudonyms.

```
t1:= id XOR c                  // XOR with secret c
if (t1 in [0, p..2^k-1]):      // If out of range...
      t2:= id                  // ...reverse if necessary
t2:= (t1 * q) mod p            // Expand with secret p
i:= t2                         // This is the exponent
b:= a^i mod p                  // The main calculation
t3:= b XOR d                   // XOR with secret d
if (t3 in [0, p.. 2^k-1]):     // If out of range...
      t3:= b                   // ...reverse if necessary
t4:= t3 << s                   // Shift-left s bits
while (t4 in [0, p.. 2^k-1]):// If out of range...
      t4:= t4 << s             // ...repeat if necessary
psdn := t4                     // This is the pseudonym
```

Figure 1. Pseudocode of the algorithm

The difficulty to find the discrete logarithm *i* of the equation a^i *mod p* is based on the assumption that *i* is randomly distributed and that no information can be used to reduce the number of possible values. This may not be the case if the persons ID is used as exponent *i*.

Two examples might help to demonstrate the problem. In both cases, *i* equals the person ID *id*. In the first example the exponent *i* is a continuous number starting with 1, so the *n*th pseudonym belongs to the person ID *n*. If an attacker is able to estimate the number of already pseudonymized persons, the number of potential *i* is heavily reduced. In the second case, the person ID is created out of the birthday and a running number (e.g. 19850323012 for the 12th person born in March 23 of 1985). Knowing that a person was born at a certain day, also limits the number of potential *i* (in the given example to 100).

To avoid the reduction of potential *i* with prior knowledge about the person ID *id*, two processing-steps are performed, including one non-linear step:

1. XOR:
   The person ID will be XORed with a constant $c \neq 0$ of *k* bits
2. EXPAND:
   The intermediate result is multiplied with an expansion factor *q* mod *p*, $(1<q<p)$

Step 1 might lead to an invalid results that is out of the range of the allowed values $(0, p..2^k\text{ -}1)$. If this happens the XOR must be reversed. In case of *p* be close to 2^k, the number of invalid values (p.. $2^k\text{-}1$) can be minimized, which lowers the risk to revers the XOR step.

p being prime guarantees that the result of step 2 is still in the range of *1..p-1*, avoiding any doubles.

At that point, even with pre-knowledge about the person ID, no conclusions about the exponent *i* of the calculation a^i *mod p* can be made, which would allow to reduce the search space. Finally, the main calculation step a^i *mod p* can be performed.

Unfortunately, if the prime number *p* is small, it is possible to calculate all possible $b = a^i \, mod \, p$ to set up a solution table $b \mapsto i$. For a prime smaller than 2^{31}, maximal 8GiB are needed to setup such a table (1GiB = 2^{30} Byte). Even for prime smaller than 2^{40}, a solution table with maximal 5TiB needs to be pre-calculated (1TiB = 2^{40} Byte). Tables with that size fit in currently used RAM or hard disks and are no burden for potential attackers. A solution to

```
t1   = id XOR c
     = 300568 XOR 1656294509 =
     = 1656593013
t2   = (t1 · q) mod p
     = (1656593013 · 41795) mod 2147483647
     = 284715408
b    = a^t2 mod p
     = 572574047^284715408 mod 2147483647
     = 465777933
t3   = b XOR d
     = 465777933 XOR 913413943
     = 766681658
t4   = t3 <<_s
     = 766681658 <<_11
     = 353489627
psdn = t4
     = 353489627
```

Figure 2. Example calculation

overcome this problem is to also keep the primitive root a secret. In that case, with given b and p, for each a a different i exists that fulfills the equation.

The entropy of the secrets a, q and c that have been used so far might be insufficient to avoid brute force attacks. So a final round of confusion is performed:

3. XOR:
   The intermediate result will be XORed with a constant $d \neq 0$ of k bits
4. LEFTSHIFT:
   The intermediate result will be shifted s bits left ($|s|>0$)

As with step 1, step 3 must be reversed, if the result is invalid. If the intermediate result of step 4 leads to an invalid value, it must be repeated until the intermediate result is in the allowed range. Both strategies do never introduce duplicates.

The calculated pseudonym *psdn* finally is the outcome of step 4. Fig. 1 on the previous page lists the entire algorithm as pseudo code.

The complexity of an attacker to re-identify the person ID is based on the secrets a, c, d, q and s and requires knowledge about some person ID / pseudonym pairs to proof if the secrets are correctly identified.

E. Example

Let $k=31$ and prime $p=2^{31}-1=2147483647$.
$a=572574047$ is a primitive root from p.
The initial value will be XORed with $c=1656294509$.
The intermediate result will be XORed with $d=913413943$.
The expansion factor is defined as $q=41795$.
Finally, an intermediate result will be shifted left with $s=11$ bits.
All calculation steps of the pseudonym for the identifier $id=300568$ are the shown in Fig. 2.
The pseudonym that has been calculated from this identifier is *353489627*.

F. Finding a primitive root

For a given prime number p it is unnecessary to find all primitive roots to select the secret a; only one primitive root is needed. The density of primitive roots is quite high so it requires approximately four random tries in case of $p=2^{31}-1$ until a primitive root is found. To proof if a selected a is a primitive root, the series of $a^i \bmod p$ $(i=1..p-1)$ has to be checked. If $a^i \bmod p = 1$ *with* $i \neq p-1$, the series can be stopped and a is not a primitive root. In that case we found two exponents resulting in the same value: $a^{i+1} \bmod p = a = a^1 \bmod p$.

The series can easily be calculated with

$$a^0 \bmod p = 1 \tag{4}$$

$$a^i \bmod p = a(a^{i-1} \bmod p) \bmod p \text{ for } i=1..p-1 \tag{5}$$

G. Calculating $a^i \bmod p$

For the calculation of $a^i \bmod p$ in the described pseudonymisation algorithm, the pre-calculation of $a^{i-1} \bmod p$ is not available; so, the recursion as mention in the previous sub-chapter is inapplicable. Alternatively the calculation can be quickened if i is split into its binary representation:

$$i = \sum_{j=0}^{k-1} 2^j \cdot i_j \quad with\ i_j \in \{0,1\} \tag{6}$$

Then

$$a^i \bmod p = \tag{7}$$

$$a^{\sum_{j=0}^{k-1} 2^j \cdot i_j} \bmod p = \tag{8}$$

$$\left(\prod_{j=0}^{k-1} a^{2^j \cdot i_j} \right) \bmod p \tag{9}$$

This calculation is very fast in case of pre-calculated $a^{2^j} \bmod p$ using

$$a^{2^0} \bmod p = a \tag{10}$$

$$a^{2^j} \bmod p = (a^{2^{j-1}} \bmod p)^2 \bmod p \quad for\ j=1..k-1. \tag{11}$$

III. Results

The algorithm for the calculation of the pseudonym would be useless, if the used secrets allow a brute-force attack. This is not the case, if the entropy of the used secretes is big enough. Furthermore, the size of the pseudonym must allow a human-readable representation and the effort to calculate the pseudonym must allow the calculation of a high number of pseudonyms per time.

A. Bit-depth of the secrets

Several secrets to calculate the pseudonym are used:

- The random number c that was used to XOR the exponent
- The factor q that was used to expand the exponent
- The primitive root a
- The random number d that was used to XOR the intermediate result
- The number of left-shifts of the intermediate result s

TABLE I. FACTS

	4-byte signed integer	5-char base64 6-char base32	2-byte signed short integer
Bits	32	30	16
maximal positive value	2^{31}-1	2^{30}-1	2^{15}-1
highest possible prime	2^{31}-1	2^{30}-35	2^{15}-19
highest possible person ID	2 147 483 646	1 073 741 789	32 748
number of invalid values	2	36	20
number of possible primitive roots of the prime	534 600 000	459 950 400	10 912

The number of possible primitive roots can be calculated with Eulers φ-function and is $\varphi(\varphi(p))= \varphi(p-1)$.

As an example, let us calculate the bit-depth of the secrets in case of data types that are usually used to store person IDs:

- 4-Byte signed integer:
  The number space is sufficient for a third of the entire living population on earth or four times the number of the living population of the European Union.
- 2-byte signed short integer
  The number space is only useful for a small set of persons, e.g., for persons of a clinical study.
- 5 chars of base64-encoded numbers or 6 chars of base32-encoded numbers
  (in case of efficient human readability)
  The number space is sufficient for two times of the living population of the European Union but insufficient for the living population the People's Republic of China.

With the information of Table I, we can calculate the entropy of the secrets that are used during the calculation (Table II).

For integer and the encoded char-values, the secret with entropy of ≈124 bits is sufficient to avoid effective brute force attacks. This is void for short integer. Here the entropy of the secrets is only ≈64 bits. In that case, the calculation of the pseudonym must be performed in two rounds with different primitive root, expansion factor, XOR and shift values. This does not fully double the entropy of the secrets because the final steps XOR and SHIFTLEFT are directly followed by another XOR step of the next round. All three steps can be simplified to only one XOR plus SHIFTLEFT. However, the entropy of the secret (≈111 bits) is sufficient today.

B. Human readability and usability

Without going into details, the readability of a pseudonym depends on the used character set plus the number of chars. Usually eight chars grouped in four chars is the maximum that a user in a day-to-day base accepts if the pseudonym has to be read and manually typed into a system. To represent the living population of the People's Republic of China at least 31 bits are needed that have to be encoded in the maximal eight chars. This allows either to use reduced character set (base32 instead of base64) or to introduce chars that are used for error correction or error detection. If a smaller population needs to be pseudonymized, the Faldum code [3] could be used. This code is able to encode 2^{30} persons in eight chars, including two chars for error detection.

TABLE II. ENTROPY OF THE SECRET

Secret	4-byte signed integer	5-char base64 6-char base32	2-byte signed short integer
q: primitive roots	≈ 29 bit	≈ 29 bit	≈ 13 bit
q: expansion factor	≈ 31 bit	≈ 30 bit	≈ 16 bit
c: XOR exponent	31 bit	30 bit	15 bit
d: XOR result	31 bit	30 bit	15 bit
s: shift result	≈ 5 bit	≈ 5 bit	≈ 4 bit
total	*≈127 bit*	*≈ 124 bit*	*≈ 63 bit*

C. Calculation speed

There are only a few steps involved in the calculation of the pseudonym. The calculation of a^i *mod p* is identified as the most time consuming calculation. The calculation is straightforward and avoids several rounds until the final result is available. Multiplications are always more time consuming than XOR or shift operations so it is assumed that the pseudonym calculation is slower that the competitive approaches. In the known scenarios, the number of pseudonymisation calculations per time is sufficient: Tests have shown that on average hardware (Intel Core 2 Duo, 2.66 GHz) 132.5-thousand pseudonyms per second can be calculated.

IV. DISCUSSION

Important for the evaluation of the algorithm is the resistance against attacks and the possibility for re-identification.

A. Attacks

It is known that for $b = a^i \bmod p$ (p prime, a primitive root of p) it is difficult to calculate the discrete logarithm i, if b, a, and p are known and p being big enough to avoid solution tables. In our case, also the primitive root a is unknown. On the other hand, there might be pre-knowledge about i. With the non-linear diffusion steps that base on the use of non-trivial secrets (e.g. $q \neq 1$, $c \neq 0$), the exponent is complex enough to make the information of the initial series useless.

Brute force attacks will only be possible if an attacker is able to validate the set of parameters with a given set of person IDs and their associated pseudonyms. An attacker will in worst case only get both sets, not knowing which person ID and pseudonym is finally linked. Depending on the size of the set it is likely, that several parameter sets lead to the same transformation of the set of person IDs to the set of pseudonyms. In case of leaked pairs of person ID plus pseudonym, this information can only be used to perform a brute force attack. A recalculation of the used parameters is not possible.

B. *Re-Identification*

A fast re-calculation of the person identifier is possible if all secrets are known. In case of small *p* and a given *a*, the solution table for $b=a^i \bmod p$ is made fast and every step of the entire calculation process can be reversed.

Only if the solution table cannot be pre-calculated, it is quicker to pseudonymise all known person IDs again to find the correct person ID.

C. *Applicability*

The concept of pseudonym-creation was implemented in a real-life scenario; in a research institute, a database with data collected for long-term studies, had to be split up into two versions: One version contains all the medical data plus the demographics of the patient as usual. The second pseudonymized version contains study specific extracts of study related data only.

To simplify the migration and to avoid an adaption of the used tools, the data-model in the pseudonymized version was kept unchanged. As a consequence, the pseudonym had to be in the same number range as the initial person ID. In our case both are integer with 2^{31}-1 as the highest possible value. Additionally, the person IDs is a consecutive running number, starting with 1.

The algorithm that was described in the paper is now used to calculate study specific pseudonyms. For each study, a different set of secrets is used as the calculation parameters. A tool was also provided to identify primitive roots.

V. CONCLUSION

The described algorithm for the creation provides a collision free one-way pseudonymisation technique for a small bit-depth of the person identifiers that allows a further use to create human readable and usable pseudonym. In contrast to other solutions, the algorithm is based on a hard problem and therefore is resistant against cryptoanalyis.

In case of more bits that are used of the person identifier, the identification of primitive roots will become more time consuming. The current strategy can only be optimized to a certain extent. It might be interesting for the future to identify the limit. But on the other hand, a pre-calculated fixed primitive root could be used for higher bit-depth, if a pre-calculation and storage of $a^i \bmod p$ is not possible. In that case one could use cloud services to find the fixed primitive root.

REFERENCES

[1] B. Alhaqbani and C. Fidge, "Privacy-preserving electronic health record linkage using pseudonym identifiers," *10th International Conference on e-health Networking, Applications and Services, HealthCom 2008*, pp. 108-117, 2008

[2] B. Riedl, V. Grascher, S. Fenz, and T. Neubauer, "Pseudonymization for improving the Privacy in E-Health Applications", *Proceedings of the 41st Annual Hawaii International Conference on System Sciences, HICSS 2008*, p. 255, 2008

[3] A. Faldum, and K. Pommerening, "An optimal code for patient identifiers," *Computer Methods and Programs in Biomedicine,* vol. 79, no. 1, pp. 81-88, 2005.

[4] J. Daemen, and V. Rijmen, *The Design of Rijndael*: Springer-Verlag New York, Inc., 2002.

[5] D. E. 3rd, and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," Request for Comments 6234, RFC 6234 (Informational), 2011.

[6] W. Diffie, and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.,* vol. 22, no. 6, pp. 644-654, 2006.

[7] S. K. Park, and K. W. Miller, "Random number generators: good ones are hard to find," *Commun. ACM,* vol. 31, no. 10, pp. 1192-1201, 1988.