The Discrete Logarithm Problem and Small Subgroup Attacks

Simon Heijungs (2625343)

February 2, 2020

1 Introduction

1.1 Background

Cryptography has two main goals. Firstly, there is confidentiality: sending messages in such a way that only the intended recipient can read them. Secondly, there is authentication: ensuring the recipient can be confident that the message comes from the right sender and has not been tempered with. The most basic way to ensure confidentiality is symmetric cryptography, where both parties share a common secret and use this to encrypt and decrypt the messages [2]. A downside of this is that it requires the involved parties to share a secret beforehand in a secure way. One solution for this is asymmetric cryptography, where the key for encryption is different from the key for decryption. With this, each party can make their own pair of a key for encryption and one for decryption and publicize the key for encryption. We call a key for encryption a public key and a key for decryption a private key.

An important concern it that this requires a way to make a pair of a public key and a private key, and it must not be possible to derive the private key from the corresponding public key. This is where one-way functions are used [2]. A one-way function is a function that is much easier to compute than its inverse. If one can design a protocol in such a way that the private key is computed from the public key using a one-way function, one can make it computationally infeasible to do the reverse.

One commonly used one-way function is modular exponentiation, using $f(x) = g^x \mod p$ where g and p are fixed integers. Its inverse, $f^{-1}(x)$ is called the discrete logarithm [2]. The theory of groups provides a convenient way to analyse those functions. A group is a set equipped with a binary operation which combines any two elements to form a third such that four conditions called "group axioms" are satisfied. A subgroup is a group formed by a subset of the elements of a (bigger) group such that combining elements in the subgroup with the operation always yields another member of the subgroup. An important class of groups that is commonly used in cryptography is the so-called "multiplicative groups of integers modulo n". Modular exponentiation can thus be studied in the context of group theory. It turns out that the existence of subgroups can make the discrete logarithm easier to solve, reducing the security of algorithms that rely on it. For this reason, modern implementations of the protocols work in a large subgroup of the original group that itself has no non-trivial subgroups. Small subgroup attacks provide a way to attack those more modern implementations. In small subgroup attacks, the attacker is one of the participants in the protocol and sends values from the wrong subgroup to trick the other party into leaking information about their private key.

1.2 Research question

This paper gives an overview of the theoretical context and aims to answer the question: *what is the relevance of subgroups for the security of cryptography based on the discrete logarithm problem?* In particular, we will focus on small subgroup attacks and defenses against those. We will mostly limit ourselves to the classical Diffie-Hellman protocol [1]. Some ideas from an algorithm called Pohlig-Hellman decomposition [7] form a basis for small subgroup attacks, so we will explain that approach as well.

The framework of this paper is represented by five articles. The most important of these is by Lim and Lee (1997) [4]. This is the original article where small subgroup attacks were first introduced. The authors concluded that small subgroup attacks form a serious problem for some protocols and they proposed several solutions. Also important is the article by Diffie and Hellman (1976) [1], which introduces the Diffie-Hellman key exchange, a protocol of which some implementations can be attacked using small subgroup attacks. The article by van Oorschot and Wiener (1996) [6] describes an attack that is a precursor to small subgroup attacks. It also proposes the idea of limiting computation to a large subgroup as a solution, which becomes important for small subgroup attacks. The article by Pohlig and Hellman (1978) [7] describes a technique known as Pohlig-Hellman decomposition, which plays a central role in the attack described by van Oorschot and Wiener [6] and the ideas it relies on are similar to the ideas used in small subgroup attacks. The article by Valenta et al. (2016) [11] gives measurements about the extent to which defenses against small subgroup attacks are actually implemented in software that is used in practice. All further articles form a theoretical background and elaboration.

1.3 Structure

In section 2, we explain the basic mathematical concepts required for understanding this paper. In section 3, we discuss the Diffie-Hellman key exchange, an important cryptographic algorithm which we will use to illustrate the attacks. In section 4, we discus the algorithms that can be used to solve the discrete logarithm problem. In section 5, we mention an approach used in many implementations of cryptographic protocols to make some of these algorithms harder compared to the work spent on the protocol. In section 6, we discus how small subgroup attacks can be used to attack these protocols. In section 7, we talk about ways in which protocols can be defended against small subgroup attacks. Section 8 concludes the paper.

2 Preliminaries

In this section we describe some preliminaries that are necessary for understanding the rest of this paper. We first explain the basics of group theory. Then, we define the discrete logarithm problem, which forms the basis of the cryptography we analyse. Finally, we discus the Chinese remainder theorem, an important mathematical result which we will need later on.

2.1 Groups and subgroups

A group G is a set S (the domain of the group) equipped with an operation $\bullet : S \times S \to S$ such that the following properties hold:

1. S is closed under $\bullet$
2. $\bullet$ is associative
3. S has an identity element 1 such that for all $x \in S$, $1 \bullet x = x \bullet 1 = x$
4. Each element $x \in S$ has an inverse $x^{-1} \in S$ such that $x \bullet x^{-1} = x^{-1} \bullet x = 1$

The number of elements in the domain S of the group is called the order of the group and is denoted $|G|$ [2].

In this article, we will focus on groups where the domain S consists of the natural numbers $1 \ldots p-1$ for some prime number p and the operation $\bullet$ is multiplication modulo p. We use standard notation for multiplication and define powers as repeated multiplication.

Example: We take $N = 11$. Then $4 \times 7 = 6$ because the usual answer 28 can be written as $2 \times 11 + 6$.

Sometimes, a subset of S equipped with the same operation $\bullet$ also forms a group. In this case, it is a subgroup of G. The order of a subgroup is always a divisor of the order of the whole group. In cyclic groups, each divisor has a corresponding subgroup of that order. Note that although we work modulo a prime p, the order of the cyclic group is $p-1$, which is not prime (except for $p = 3$), because 0 is not part of the group.

When you have an element $g \in S$ and take the set of all elements that can be written as g^x for some integer x, those elements form a subgroup or the whole group. We say that g is a generator for this subgroup or group. Each cyclic group has at least one generator.

2.2 The discrete logarithm problem

The discrete logarithm is similar to the normal logarithm except that it is defined for groups: If we have $g^k = r$ with g and r being elements of a group and k

being an integer, then $\log_g r = k$. Exponentiation in groups is computationally easy, while the discrete logarithm is computationally hard in the general case. It is therefore a one-way function, which makes it useful for cryptography.

2.3 The Chinese remainder theorem

The Chinese remainder theorem is an important result in modular arithmetic. It states the following:

Given integers $n_1, \ldots, n_l$ that are pairwise relative prime (having no divisors > 1 in common), and integers $a_1 \ldots a_l$ such that $a_i < n_i$ for all i, then the system of equations

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ &\ldots \\ x &\equiv a_l \pmod{n_l} \end{aligned}$$

has a solution that is unique modulo $\prod_{i=1}^{l} n_i$. This solution can algorithmically be found in an efficient manner [3].

3 Diffie-Hellman key exchange

In this section, we describe the Diffie-Hellman key exchange protocol, which is an important cryptographic application of the one-way nature of modular exponentiation. The Diffie-Hellman key exchange [1] is a protocol for getting two parties to agree on a common key using only an insecure network such that an eavesdropper cannot find out this key. While there are some modern variations that work a bit differently 4, a classical Diffie-Hellman key exchange between Alice and Bob works as follows:

1. Alice and Bob agree on a large prime p and a generator g for the multiplicative group modulo p. All further numbers in the protocol are modulo p.
2. Alice generates a random integer k_A and computes $r_A = g^{k_A}$. She keeps k_A secret and sends r_A to Bob.
3. Bob generates a random integer k_B and computes $r_B = g^{k_B}$. He keeps k_B secret and sends r_B to Alice.
4. Alice computes the key $K = r_B^{k_A}$ and Bob computes $K = r_A^{k_B}$. Since $(g^{r_A})^{r_B} = g^{r_A r_B} = (g^{r_B})^{r_A}$ they get the same key K.

This protocol is illustrated in figure 1.

An eavesdropper Eve can get her hands on r_A, r_B, g and N but computing k_A or k_B from there is an instance of the discrete logarithm problem. It is widely believed that solving the discrete logarithm problem is the only way Eve can compute K, although this has not been rigorously proven. 2

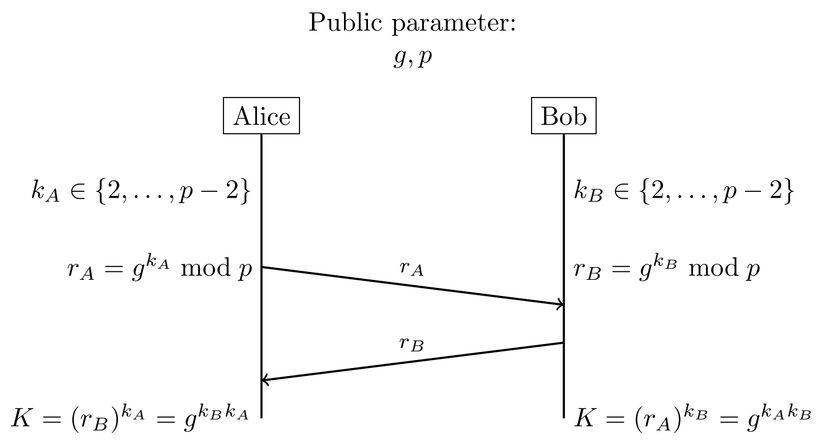
Public parameter:
g, p
Alice
Bob
$k_A \in \{2, \ldots, p-2\}$
$k_B \in \{2, \ldots, p-2\}$
$r_A = g^{k_A} \bmod p$
$r_B = g^{k_B} \bmod p$
r_A
r_B
$K = (r_B)^{k_A} = g^{k_B k_A}$
$K = (r_A)^{k_B} = g^{k_A k_B}$

Figure 1: The Diffie-Hellman key exchange (based on code from `https://www.iacr.org/authors/tikz/`)

An important caveat is that the Diffie-Hellman protocol is not by itself secure against an attacker who can alter the messages. Such an attacker is known as a "man in the middle". An attacker can generate his own k_A and k_B, and use k_A to follow the protocol with Bob and k_B to follow the protocol with Alice. Now the attacker has a key with Alice and a different one with Bob, so he can decrypt, re-encrypt and relay every message. To prevent this, authentication is needed. The purpose of the Diffie-Hellman protocol is to a different key K for each session of communication which are discarded afterwards, so even an attacker who obtains the keys for authentication cannot decrypt messages from the past. Methods for providing authentication are beyond the scope of this paper.

4 Algorithms for solving the discrete logarithm problem

In this section, we describe several algorithms that have been developed for finding the discrete logarithm. We start with the naive approach. Then, we describe some general ways to improve the performance. Finally, we explain Pohlig-Hellman decomposition, a further improvement that uses any subgroups that exist to reduce the problem size. This technique also forms the basis for small subgroup attacks, which are the main focus of this paper.

4.1 Exhaustive search

A simple approach to solving the discrete logarithm problem $\log_g r$ is to keep computing g^k for different values of k until one is found for which the result is r [10]. This is called exhaustive search. On average, it takes about $\frac{|G|}{2}$ attempts. In cryptography, this number would generally have hundreds of digits, making this approach infeasible.

4.2 $O(\sqrt{|G|})$ algorithms

There are faster ways to compute the discrete logarithm problem. The more efficient algorithms tend to have $O\left(\sqrt{|G|}\right)$ running time. We will explain the baby-step giant-step algorithm here and only mention a few other algorithms.

4.2.1 Baby-step giant-step

The baby-step giant-step algorithm [10] makes a trade-off between time and memory. When solving $g^k = r \bmod N$, let $m = \left\lceil \sqrt{|G|} \right\rceil$. Write k as $k_1 m + k_2$ where k_1 and k_2 are both non-negative integers $\leq m$. Now one can reason as follows:

$$\begin{aligned} g^k &= r \\ g^{k_1 m + k_2} &= r \\ g^{k_1 m} g^{k_2} &= r \\ g^{k_2} &= r g^{-k_1 m} \\ g^{k_2} &= r \left(g^{-m}\right)^{k_1} \end{aligned}$$

Precompute all m values of g^{k_2} and save them in a hash table along with the corresponding value of k_2. Then, for each value of k_1, compute $r\left(g^{-m}\right)^{k_1}$ and look up the result in the hash table. If you find a match, compute k as $k_1 m + k_2$.

Each value of g^{k_2} takes constant time to compute and there are at most m of those, so this takes $O(m) = O\left(\sqrt{|G|}\right)$ work. Since lookups in a hash table take pretty much constant time, $r\left(g^{-m}\right)^{k_1}$ takes constant time as well, so doing that at most m times takes $O\left(\sqrt{|G|}\right)$ time as well. Therefore, the overall time complexity is $O\left(\sqrt{|G|}\right)$.

4.2.2 Other algorithms

Pollard's rho algorithm [10] and Pollard's kangaroo [10] algorithm are a lot more difficult to describe concisely and have about the same running time as baby-step giant-step. Pollard's rho algorithm has the benefit that it takes little memory, while baby-step giant-step takes $O\left(\sqrt{|G|}\right)$ memory. Both of Pollard's algorithms are Monte Carlo algorithms that rely on a hash function h from the group G to a range of natural numbers to make a sequence S where $S_{n+1} = S_n \times g^{h(S_n)}$. They use the fact that this sequence eventually becomes cyclical to derive a relation between elements.

Pollard's kangaroo algorithm (also known as Pollard's lambda algorithm) has the benefit that when you know in advance that the result will be in some interval of size s, the running time is only $O\left(\sqrt{s}\right)$.

Pollard 9 also developed a very advanced algorithm called the number field sieve algorithm, which will not be explained here. This is currently the fastest known algorithm for very large instances of the problem. It also has the advantage that a large part of the work only depends on the order of the group, making it even more efficient when multiple discrete logarithms in the same group have to be computed.

4.3 Pohlig-Hellman decomposition

Pohlig-Hellman decomposition [7] is a technique to split up a large discrete logarithm problem into several smaller ones, the sizes of which correspond to the prime factors of the order of the original one. The idea is to solve discrete logarithms in subgroups of the original group, whose orders correspond to the prime factors, and find a modulus of the final answer for each subgroup. Those moduli can then be combined using the Chinese remainder theorem. One can apply it to solve $g^k = r$ as follows:

First, factorize the order of the group $|G|$ as $\prod_{i=1}^{l} p_i^{e_i}$ where p_i are the prime factors of $|G|$, e_i are their corresponding multiplicities, and l is the number of unique prime factors.

Then, for each unique prime factor p_i, compute $k_i = k \mod p_i^{e_i}$ using the following steps:

1. Compute S_0 through S_{p_i-1} by $S_x = g^{x|G|/p_i}$
2. Write k_i as $\sum_{j=0}^{e_i-1} d_j p_i^j$ where $0 \leq d_j \leq p_i - 1$
3. Let $b_0 = a$
4. Compute each d_j starting with d_0 as follows:
   (a) Compute $b_j^{(p-1)/p_i^j}$ and search it in r_0 through r_{p_i-1}. d_j is the index where it is found.
   (b) Let $b_{j+1} = b_j g^{-p_i^j d_j}$

Finally, combine the obtained moduli k_i using the Chinese remainder theorem. Figure 2 gives an illustration of the workflow of this algorithm.

One can also perform a partial Pohlig-Hellman decomposition where only the small prime factors are used. Then, if s bits of the solution are still unknown, they can be obtained in $O(\sqrt{s})$ time using Pollard's kangaroo algorithm [6].

$$\begin{tikzcd}
& \begin{array}{c} r = g^k \pmod{|G|} \\ |G| = p_1 \times \cdots \times p_l \end{array} \arrow[dl, bend right] \arrow[dr, bend left] \arrow[d, phantom, "\text{Pohlig-Hellman}"] & \\
k \pmod{p_1} \arrow[dr] & \dots \arrow[d, phantom, "\text{CRT}"] & k \pmod{p_k} \arrow[dl] \\
& k \pmod{|G|} &
\end{tikzcd}$$

Figure 2: How Pohlig-Hellman decomposition can be used (based on code from `https://www.iacr.org/authors/tikz/`)

5 Limiting computation to a large subgroup

Since Pohlig-Hellman decomposition allows an attacker to learn a number of bits from the key based on small subgroups, this makes the space in which an attack has to search for the key smaller. A simple solution would be to increase the length of the exponents, and thus the search space the attacker has to search, to compensate for this. A downside of this approach is that it makes the protocol substantially slower. A more common approach is to choose a generator that only generates a large subgroup of size q.

6 Small subgroup attacks

As we discussed, the Diffie-Hellman key exchange on its own lacks authentication, which makes man-in-the-middle attacks possible. Because of this, many protocols based on the principle of Diffie-Hellman add public-private key pairs that are also used in the protocol.

These public-private key pairs can also be made using the discrete logarithm problem, making it easy to integrate them into the calculations. This is done by making a private key y and a public key $y = g^x$. One such protocol is the authenticated MTI (Matsumoto-Takashima-Imai) protocol. Here, each party also has a public identifier I. The following is an outline of this protocol:

1. Alice randomly picks $k_A \in \mathbb{Z}_q$, computes $r_A = g^{k_A} \mod p$ and sends r_A to Bob.

2. Bob randomly picks $k_B \in \mathbb{Z}_q$, computes $r_B = g^{k_B} \mod p$, $K_B = y_A^{k_B} r_A^{x_B}$, $e_B = h(K_B, r_B, r_A, I_B, I_A)$, and sends $\{r_B, e_B\}$ to Alice.

3. Alice computes $K_A = y_B^{k_A} r_B^{x_A}$ and $e'_B = h(K_A, r_B, r_A, I_B, I_A)$, and checks that $e_B = e'_B$. If $e_B \neq e'_B$, then Alice stops the protocol with failure. (Optional) Otherwise, Alice computes $e_A = h(K_A, r_A, r_B, I_A, I_B)$ and sends e_A to Bob.

4. (Optional) Bob computes $e'_A = h(K_B, r_A, r_B, I_A, I_B)$, and checks that $e_A = e'_A$. If $e_A \neq e'_A$, then Bob stops the protocol with failure.

Here, g is a generator of a large subgroup of order q as we discussed in section 5. The shared key the parties will use is $K_A = k_B$. This equality holds because $y_A^{k_B} = (g^{x_A})^{k_B} = (g^{k_B})^{x_A} = r_B^{x_A}$ and likewise, $y_B^{k_A} = r_A^{x_B}$. This equality will only hold if both parties use their correct private keys, which an attacker will not have. e_A and e_B are generated and compared to verify that equality in a safe way.

The protocol is still not necessarily secure, though. If no extra checks are done, this protocol is vulnerable to small subgroup confinement attacks.

According to the protocol, Alice has to compute r_A as g_A^k in step 1, necessarily making it an element of the large subgroup of g, but this is not enforced. In particular, Alice can send a value for r_A to Bob which generates a subgroup that is small enough to search exhaustively, say size s. If she does that, the computation of K_B by Bob in step 2 becomes more predictable. The first factor $y_A^{k_B}$ can always be computed as $y_A^{k_B} = (g^{x_A})^{k_B} = (g^{k_B})^{x_A} = r_B^{x_A}$. The second factor, $r_A^{x_b}$ is an element of the group generated by r_a, so there are only s possibilities. e_B is calculated based on K_B, R_B, R_A, I_B and I_A. Alice knows all of those values except K_B, so e_B is equally predicable. Alice can now try values from g^0 to g^s until she finds one that matches what Bob sent. If that value has exponent v, she can conclude that $x_B \equiv v \mod s$. Alice can now stop the protocol and do the whole thing again with a few different small subgroups to get different moduli. She can then combine all the moduli using the Chinese remainder theorem to narrow x_B down to either one value or a small enough set such that she can try all possibilities.

This is one example of a small subgroup confinement attack. The essential part is that Bob computes c^{x_b} where Alice can choose c, and uses this to derive something he shares with Alice. This occurs in some other protocols as well, and similar attacks can be done there.

There are a few other attacks based on small subgroups as well. One is small subgroup non-confinement attacks. Those abuse the fact that some protocols can in rare cases land in small subgroups and retry in those cases. If this does not happen when an attacker guesses a key, the attacker can rule out a few other possible keys than the one she tried. [4]

7 Mitigations

To prevent small subgroup attacks, several mitigations are possible.

The most direct one is to check each received value to verify that it is in the agreed large subgroup. This can be done using the following relation: if the agreed large subgroup has order q, an element e is in that subgroup iff $e^q = 1$. A downside of this approach is that those extra checks make the protocols substantially slower.

Another way to mitigate small subgroup attacks is to use a so-called safe prime. A safe prime p is a prime such that $\frac{p-1}{2}$ is prime as well. If such a prime is used, the only small subgroup is $\{1, p-1\}$ of order 2. This in itself limits small subgroup attacks so they can only find one bit of the key: the parity bit. This can also easily be prevented because this subgroup is much easier to check for than other subgroups. The main problem here is that safe primes are quite rare and take longer to find.

Finally, one can also use elliptic curves instead of multiplication modulo a prime to generate the groups. The main problem we had with groups generated with multiplication modulo a prime p is that the order, $p-1$, is not a prime so our groups had subgroups that an attacker can use. Elliptic curves can generate groups of prime order, so there are no subgroups to begin with. This means attackers simply have nothing to work with for Pohlig-Hellman decomposition or small subgroup attacks.

Groups based on elliptic curves are used extensively in practice nowadays. The danger of small subgroup attacks is one of several reasons for this. That being said, groups based on multiplication modulo a prime are still used a lot as well. Some standards for this are actually incompatible with the use of safe primes. This implies that the checks of the first mitigation are necessary. A study published in 2017 found that out of the 20 cryptographic libraries the authors examined, none performed those checks by default. They also found that less than 3% of the servers following the standards they examined performed those checks. 11

8 Conclusion

Many cryptographic algorithms rely on the difficulty of the discrete logarithm problem. While the hardest instances of the problem still appear to be difficult enough, there are also much easier instances on which we cannot securely rely. In particular, in instances of the problem where the group has many small subgroups, Pohlig-Hellman decomposition can make computing the discrete logarithm much easier. This makes it important to ensure that you rely on a hard instance of the problem when implementing cryptographic protocols.

When working in a group that is a subgroup of a larger group, one should be wary about using values received from another party in conjunction with long-term secret values. If the received value is not in the subgroup where it should be, you can leak parts of a secret this way.

The best solution is probably to switch to groups defined by elliptic curves. They can be made with a prime order, so they bypass the problem of subgroups altogether. If this is not possible, additional checks need to be made to enforce the choice of subgroup. Elliptic curve cryptography is already becoming widely adopted in practice, but there are still applications that use groups defined by modular multiplication. We need to pay extra attention to ensure that those implementations mitigate against small subgroup attacks.

References

1. Diffie W., Hellman, M.E. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory* 22(6): 644-654. doi:10.1109/TIT.1976.1055638

2. Hoffstein J. et al. *A Mathematical Introduction to Cryptography.* Second edition. Springer, New York, NY.

3. Knuth D.E. (1998). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Third edition. Addison-Wesley, Boston, MA.

4. Lim C.H., Lee P.J. (1997). A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup. *Advances in Cryptology - CRYPTO '97. Lecture Notes in Computer Science* 1294: 249-263. doi:10.1007/BFb0052240.

5. Lipschutz S., Lipson M.L. (2007). *Schaum's Outline of Discrete Mathematics.* Third edition. McGraw-Hill, New York, NY.

6. van Oorschot P.C., Wiener M.J. (1996). On Diffie-Hellman Key Agreement with Short Exponents. In: Maurer U. (eds) *Advances in Cryptology EUROCRYPT 96.* EUROCRYPT 1996. Lecture Notes in Computer Science, vol 1070. Springer, Berlin, Germany.

7. Pohlig S.C., Hellman M.E. (1978). An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance. *IEEE Transactions on Information Theory* 24: 106-110. doi:10.1109/TIT.1978.1055817

8. Pollard J.M. (1978). Monte Carlo Methods for Index Computation $\pmod P$. *Mathematics of Computation* 32(143): 918-924. doi:10.2307/2006496.

9. Pollard J.M. (1993). Factoring with cubic integers. In: Lenstra A.K., Lenstra H.W. Jr. (eds), *The Development of the Number Field Sieve* Springer, Berlin, Germany.

10. Shoup V. (2005). *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, Cambridge, UK.

11. Valenta, L. et al. (2017). Measuring Small Subgroup Attacks Against Diffie-Hellman. *Network and Distributed System Security Symposium* doi: 10.14722/ndss.2017.23171