

# Subversion in Practice: How to Efficiently Undermine Signatures

Joonsang Baek<sup>1</sup>, Willy Susilo<sup>1</sup>, Jongkil Kim<sup>1</sup>, and Yang-Wai Chow<sup>1</sup>

<sup>1</sup>University of Wollongong, Australia

December 13, 2018

## Abstract

Algorithm substitution attack (ASA) on signatures should be treated seriously as the authentication services of numerous systems and applications rely on signature schemes and compromising them has a significant impact on the security of users. We present a somewhat alarming result in this regard: a highly efficient ASA on the Digital Signature Algorithm (DSA) and its implementation. Compared with the generic ASAs on signature schemes proposed in the literature, our attack provides fast and undetectable subversion, which will extract the user's private signing key by collecting maximum three signatures arbitrarily. Moreover, our ASA is proven to be robust against state reset. We implemented the proposed ASA by replacing the original DSA in `Libgcrypt` (a popular cryptographic library used in many applications) with our subverted DSA. Experiment shows that the user's private key can readily be recovered once the subverted DSA is used to sign messages. In our implementation, various measures have been considered to significantly reduce the possibility of detection through comparing the running time of the original DSA and the subverted one (i.e. timing analysis). To our knowledge, this is the first implementation of ASA in practice, which shows that ASA is a real threat rather than only a theoretical speculation.

## 1 Introduction

*Algorithm Substitution Attack (ASA).* We trust cryptography expecting that it will provide a maximum level of security. If cryptography turns out to be subverted so that it leaks the critical information of a user to the third party surreptitiously, we would seriously be concerned about its impact on security and privacy. For this reason, many were surprised and even shocked by Snowden's revelation that the subversion of certain cryptographic algorithms really happened [13]. In response to this, the research community investigated subversion of cryptography closely and Bellare, Paterson and Rogaway came up with the formal notion called "algorithm substitution attack (ASA)" [6]. Roughly, the ASA enables an adversary, often called "big brother", to replace some parts of a legitimate cryptographic scheme or protocol with a modified one in order that it subliminally leaks the crucial information about a secret key or a confidential message to the adversary.

*ASA on signatures.* The focus of this paper is ASA on signature schemes. One of the possible real-world *scenarios* can be described as follows: It is common nowadays that a crypto library is embedded as a subcomponent of commercial software or hardware, which is installed and patched by the vendor. A problem is that if the vendor is malicious or enforced to act maliciously by an external party, it can easily replace the regular crypto library with a subverted one to obtain the victim's private key. If the subverted library does not require any hidden communication channel nor result in degradation

of performance, the victim cannot detect any difference and will be deceived into using the subverted library.

Although “kleptography” proposed by Young and Yung dealt with the subversion of signature schemes [30], the first formal treatment of the subject matter in the context of ASA was made by Ateniese et al [2]. Their subverted signature scheme is generic but inefficient. A more efficient ASA on a certain group of signatures was proposed very recently by Liu et al [16]. (Note that these related works will be discussed in greater detail in the next section.) However, there still remain some important problems to address, which we discuss in what follows.

Realizing ASA. We came to the realization that none of the ASAs proposed in the literature has been implemented although the original motivation for ASA is closely related to the practice of cryptographic algorithms in the real world. Moreover, discovering ASAs on widely-used cryptographic algorithms is important as they are already deployed in numerous applications and their ASA vulnerability has deep impact on the security of those applications. These have motivated us to pursue research with two directions: how to come up with practical ASAs on known cryptographic schemes and how to realize them efficiently.

State reset. According to [6], subverted cryptographic schemes are classified into stateful and stateless ones. As stateless schemes do not maintain state, there is no need to concern about *state resets*, which usually happen with rebooting or cloning to create a virtual machine. They, however, can lead to detection when it comes to stateful schemes. Nevertheless, state reset has not been captured by the definition of detectability for ASA on signature schemes, previously considered in the literature. Our detectability notion for stateful subverted signature schemes includes state reset and our subverted schemes are analyzed based on it.

Our contributions. We present an efficient ASA on the DSA [10], which we sometimes refer to as a “subverted DSA scheme”. As follows is a summary of our results:

- By making use of the randomized component of a signature contiguous to the current one as a source of randomness, we were able to design a conceptually simple yet highly efficient subversion on the DSA: It only takes a maximum of *any* three consecutive signatures to extract a private signing key, which makes our subversion remarkably practical.
- We refined the formal definition of the undetectability for ASA on signatures so that it can incorporate state reset. We proved that our subverted DSA is *undetectable* and *robust against state reset*. To our knowledge, this is the first work dealing with state reset for ASA on signatures *formally*.
- We implemented the proposed ASA by substituting the original DSA provided in the `Libgcrypt` library [12] with our subverted DSA. Our implementation carefully considered countermeasures against the possible detection of subversion by comparing the running time of the subverted signature with that of the original DSA.

On Generalization. We are aware that the proposed subversion technique can be generalized to cover the signature schemes that have a structure similar to the DSA, such as Schnorr and the ECDSA. (We did show in this paper that our ASA can be applied to the Schnorr signature [24].) However, we do not present a complete specification of generalization in this version of the paper as its focus is an efficient realization of ASA on the mentioned signature schemes rather than a theoretical generalization, which may result in some redundancy.

Organization. The rest of this paper is organized as follows. In the following section, we review and discuss the related work in detail. In Section 3, we provide definitions of key extraction and undetectability for subverted signature schemes. In Section 4, we present our subverted DSA scheme. Our implementation of the subverted scheme and its analysis are presented in Section 5. Concluding

remarks will follow in Section 6. As an additional instance of the proposed ASA, we present an ASA against the Schnorr signature scheme in Appendix .1.

## 2 Related Work

The consequences of ASA on signature schemes can be catastrophic [2]. – The attacker’s goal in this case may be to forge signatures on particular messages. Perhaps his intention goes beyond this and may want to steal signers’ private keys, e.g., to impersonate individuals or to forge certificates. In this section, we elaborate on the previous work on subverting signature schemes.

*Early work.* Young and Yung pioneered research on ASA, which they called “kleptography” [29]. In their series of work [29][30][31][32], they presented kleptographic attacks, described as SETUP (Secretly Embedded Trapdoor with Universal Protection) against various cryptographic schemes including RSA encryption, Diffie-Hellman key exchange, Schnorr, ElGamal and DSA signatures. Our ASA is inspired by their work, especially through exploiting the arithmetic structure of discrete-logarithm based signatures to extract a private key. (More discussion on this will follow in the beginning of Section 4.) Their basic idea is to exploit the structural vulnerability of many discrete-logarithm based signatures. – Security problems can arise in those signatures if a non-uniform value for an ephemeral component (nonce) in the signature is used [31][32]. This exploit has been known for a long time. Notable results include Simmons’ seminal work on subliminal channel [25][26] and Nguyen and Shparlinski’s attack on DSA using partially known nonces [19].

The kleptographic attacks from [30] and [32] require the adversary to locate *one* particular subverted signature (possibly from many) to extract a private key. To achieve this, the subverted scheme needs to maintain state to “flag” such a signature and transmit it to the adversary. We envision that in practice, this is harder than it looks since there are many subverted signatures that the victim will have generated and the adversary has to closely monitor them to locate one particular signature.

We also point out that their attack uses a deterministic subversion key, which can lead to detection of ASA if a *state reset* occurs through a reboot or cloning a virtual machine [6][4]. In other words, the same signature can be outputted twice in the case of a state reset, which renders the attack detectable as the non-subverted scheme should be randomized.

*Ateniese et al.’s work.* Ateniese et al. presented two generic ASAs on any randomized signature schemes [2]. Their first attack is based on Bellare et al.’s ASA technique [6], whereby the adversary can extract each bit of the secret key from a vector of ciphertexts generated by a subverted encryption algorithm using biased randomness. Ateniese et al. applied this technique to mount ASA on signature schemes which possess sufficient randomness (i.e. the signature schemes that contain as many random bits as the size of the secret signing key). The second attack that Ateniese et al. proposed is applicable to a group of signature schemes which have little randomness.

Although Ateniese et al.’s ASAs can be applied to a large class of signature schemes, compared with theirs, our ASA on discrete-logarithm based signature schemes has the following merits.

- (1) The adversary needs to collect (much) fewer subverted signatures to recover a signing key: Even though Ateniese et al.’s first attack applies to generic randomized signature schemes, the adversary needs to collect as many as  $|sk|$  signatures (where  $|sk|$  is the bit-length of a private key). Their second attack, applicable only to “coin extractable” signature schemes which exclude DSA and Schnorr schemes, requires  $\ell/d$  signatures to be collected, where  $2^d$  is the size of randomness space and  $\ell$  is the size of the signing key. (Note that in the coin extractable signature schemes, the randomness used in the signature generation is publicly available.) On the other hand, our ASA requires only a maximum of any three consecutive signatures to ensure the successful recovery of a private signing key.

- (2) State is managed much more efficiently in our ASA: The above two attacks need to maintain state, i.e., they are stateful. In the first attack, the state needs to locate at least the first and last signatures (among the collected signatures), which will be used to exfiltrate the first and last bit of a private key respectively. Similarly, the state of the second attack needs to locate the first and last signatures, which will be used to extract the first and last chunks of bits of a private key respectively. Although Ateniese et al. [2] claim that their attack can be made stateless by providing the state as a message to the subverted signing algorithm, we point out that this is only possible when the adversary has control over inputs to the subverted signing algorithm, which may not always be the case in practice. In contrast, our ASA does not require this assumption. Even though the state should also be maintained in our ASA (i.e., our ASA is also stateful), its purpose is not to locate or flag (subverted) signatures to extract signing keys but rather to perform alternate operations for subverted signature generation.
- (3) Moreover, our subverted signature scheme is proven to resist the state reset while it is not considered in Ateniese et al.’s schemes. – We suspect that achieving provable robustness against state reset seems hard as their subversion methods are based on Bellare et al.’s subversion method which exploits the “coin-injectiveness” that cannot be shown to resist the state reset as discussed in [6].

*Liu et al.’s work.* Liu et al. recently proposed a generic ASA that improves Ateniese et al.’s ASA on signature schemes [16]. Although their ASA uses a similar idea to divide the subverted signing algorithm into two parts to boost efficiency of the attack, we point out the following difference and a flaw in their argument.

1. The key difference between Liu et al.’s work and ours is that our ASA is symmetric subversion [6][2][4] while their is asymmetric one, which was proposed by Young and Yung [30][32] to prevent the loss of a subversion key. (That is, by generating a subversion public and private key pair and embedding the public key only in the subverted algorithm, the adversary can conduct subversion more securely as the embedded public key does not reveal anything about its matching private key.) However, what we realized is that the asymmetric subversion is prone to “timing analysis” since it will usually involve heavy public operations in the subverted algorithm and make the subverted algorithm run slower than the original one. Our experiment shows this is the case as presented in Section 5. On the other hand, our cryptomalware based on symmetric subversion is resilient to the timing analysis as compared with Liu et al.’s.
2. Liu et al. makes a wrong claim that their attack is stateless. In their subverted signature scheme, the randomness used in the current signature generation depends on the randomness used to generate a signature previously, which clearly shows that their subversion is stateful.
3. As is with the case of Ateniese et al.’s work, the security against a state reset was not considered in Liu et al.’s work.

*Other related work.* Apart from signatures, active research on ASA against encryption and its prevention has been conducted since Bellare et al.’s seminal work [6]. Various types of ASAs on symmetric encryption schemes have been proposed by Bellare et al. [6][4]. In particular, the ASA presented in [4] is a stateless attack, which removes the limitation of stateful ASAs presented in [6].

As a prevention measure for ASA on symmetric encryption, the works [3][4][6] recommended deterministic algorithms. Bellare and Hoang extended the result to public key encryption [3]. (Ateniese et al. also proposed a deterministic solution for ASA against signatures [2].) Others developed somewhat different approaches including the cryptographic reverse firewall (RF), which resides between the user’s computer and the outside environment to protect cryptographic schemes and protocols from

insider attacks [17]. More recently Russell et al. proposed a generic technique for designing public key encryption that is semantically secure against ASA, which utilizes inter-component checking techniques with offline component [21].

Though a number of attempts have been made to guard against ASAs, each solution has its own merits and shortcomings. Hence, we can cautiously state that no perfect solution for ASA has emerged yet.

### 3 Notions of Subverted Signature

Syntax for subverted signature schemes. We now provide the precise syntax for a subverted signature scheme.

**Definition 1.** Let  $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Verify})$  be a regular (non-subverted) signature scheme. The  $\text{Gen}$  algorithm takes a security parameter  $\lambda$  as input and produces a private and public key pair  $(sk, pk)$ . The  $\text{Sign}$  algorithm generates a signature  $\sigma$  on a message  $m$ . The  $\text{Verify}$  algorithm checks whether a given signature  $\sigma$  on the message  $m$  is valid or not, outputting 1 or 0 respectively. A subverted signature scheme for  $\text{SIG}$ , which we denote by  $\overline{\text{SIG}} = (\overline{\text{Gen}}, \overline{\text{Sign}})$  is defined as follows.

- The subversion key generation algorithm  $subk \leftarrow \overline{\text{Gen}}(1^\ell)$ : On input a security parameter  $\ell$ , this algorithm generates a subversion key  $subk$ .
- The subverted signing algorithm  $(\bar{\sigma}, \iota) \leftarrow \overline{\text{Sign}}(subk, sk, m, \iota)$ : On input a subversion key  $subk$ , a private key  $sk$ , a message  $m$  and a state  $\iota$ , this algorithm produces a subverted signature  $\bar{\sigma}$  on the message  $m$  with an updated state  $\iota$ .

We require that the subverted signature  $\bar{\sigma}$  (on the message  $m$ ) can be verified in the same way as the non-subverted signature scheme is verified, i.e.,  $1/0 \leftarrow \text{Verify}(pk, \bar{\sigma}, m)$ .

The above definition is similar to Ateniese et al.'s [2] but we have made the generation of a subversion key and its presence as input to a subverted signing algorithm more explicit (rather than assuming the presence of “hard-wired arbitrary auxiliary information“ as described in [2]), following the description style of Bellare et al.'s work in [6] and [4].

Note that in the definition above the subverted signing algorithm is defined as stateful by default, taking state  $\iota$  as input and returning the updated state. (In our subverted schemes which will be presented later, state is only used internally and is never returned externally by the signing algorithm.) Key extraction. The most serious attack against a signature scheme may be the private key extraction. Usually extracting a private signing key from regular signature schemes is deemed infeasible but this is not the case when it comes to a subverted signature scheme. – The adversary (Big Brother) should be able to recover the private signing key efficiently from the number of subverted signatures obtained. In the formal definition below, we formulate the key extraction in the subverted signature scheme.

**Definition 2.** Let  $\mathcal{B}$  be an adversary. Let  $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Verify})$  be a signature scheme. Suppose that  $\overline{\text{SIG}} = (\overline{\text{Gen}}, \overline{\text{Sign}})$  is a subverted signature scheme for  $\text{SIG}$ . Let  $Q_S$  be the number of signing queries. Consider the following game:

**KeyExtract** $_{\mathcal{B}, \text{SIG}, \overline{\text{SIG}}}(\lambda)$

1. Run  $\text{Gen}(1^\lambda)$  to generate  $(pk, sk)$ .
2. Run  $\overline{\text{Gen}}(1^\ell)$  to generate  $subk$ .
3. For  $i = 1, \dots, Q_S$ , do the following:

If  $\mathcal{B}$  queries each message  $m_i$ , compute  $(\bar{\sigma}_i, \iota_i) \leftarrow \overline{\text{Sign}}(subk, sk, m_i, \iota_{i-1})$  and return  $(\bar{\sigma}_i, \iota_i)$  to  $\mathcal{B}$ .

4.  $\mathcal{B}(pk, subk, \{\bar{\sigma}_1, \dots, \bar{\sigma}_{Q_S}\})$  outputs  $sk'$ .
5. If  $sk' = sk$ , return 1. Otherwise return 0.

We define  $\text{Adv}_{\mathcal{B}, \text{SIG}, \overline{\text{SIG}}}^{\text{extract}}(\lambda) = \Pr[\mathbf{KeyExtract}_{\mathcal{B}, \text{SIG}, \overline{\text{SIG}}}(\lambda) = 1]$ , where the probability is taken over the random coins used by  $\mathcal{B}$ , as well as the random coins used in the game.

We remark that the notion of key extraction is not formally defined in [2], even though their attack implies it. Another remark is that the key extraction is *not* mandated as a security goal but as an attack goal, which is a reasonable requirement for subversion as the similar notion for encryption is defined in [4].

*Undetectability under state reset.* We now define what it means for a subverted signature scheme to be secure from the adversary's (Big Brother's) point of view. Intuitively, an ordinary user, who does not possess the adversary's subversion key, should not be able to *detect* that a signature has been created by a subverted signing algorithm. On the other hand, it is natural to assume that the user, who is trying to detect subversion, has access to the private signing key since the user of a concerned signature scheme (not an outsider) is usually more interested in detecting subversion. Therefore, in the following definition of undetectability, a user (any party that is interested in detecting subversion) is given the private key  $sk$  that corresponds to a public key  $pk$ .

Another aspect one can consider in the definition of undetectability is state reset [6][4]. This can be achieved by giving the user access to a reset oracle. – When this oracle is invoked, state used in the subverted signature scheme is initialized (reset). State reset obviously makes the user more powerful but it was not considered in the previous undetectability notions of subverted signatures presented [2] and [16].

As follows is our formal definition of undetectability.

**Definition 3.** Let  $\mathcal{A}$  be a user (represented as an algorithm). Let  $\text{SIG} = (\text{Gen}, \text{Sign}, \text{Verify})$  be a signature scheme. For a subverted signature scheme  $\overline{\text{SIG}} = (\overline{\text{Gen}}, \overline{\text{Sign}})$ , consider the following game:

**Detect** $_{\mathcal{A}, \text{SIG}, \overline{\text{SIG}}}(\lambda)$

1. Run  $\text{Gen}(1^\lambda)$  to generate  $(sk, pk)$ .
2. Run  $\overline{\text{Gen}}(1^\ell)$  to generate  $subk$ .
3.  $b \in \{0, 1\}$  is chosen at random.
4.  $\mathcal{A}^{\text{SignO}_b(\cdot), \text{Reset}(\cdot)}(sk, pk)$  outputs its guess  $b' \in \{0, 1\}$ .
5. If  $b' = b$ , return 1. Otherwise return 0.

-----

$\text{SignO}_b(m_i)$

If  $b = 0$ ,  $\sigma'_i \leftarrow \text{Sign}(sk, m_i)$   
 Else  $(\sigma'_i, \iota_i) \leftarrow \overline{\text{Sign}}(subk, sk, m_i, \iota_{i-1})$   
 Return  $\sigma'_i$

-----

$\text{Reset}(rt_i)$  //  $rt_i$  is a reset query

$\iota_i \leftarrow \emptyset$

-----

We define  $\text{Adv}_{\mathcal{A}, \text{SIG}, \overline{\text{SIG}}}^{\text{detect}}(\lambda) = |\Pr[\mathbf{Detect}_{\mathcal{A}, \text{SIG}, \overline{\text{SIG}}}(\lambda) = 1] - \frac{1}{2}|$ , where the probability is taken over the random coins used by  $\mathcal{A}$ , as well as the random coins used in the game. We say that  $\overline{\text{SIG}}$  is undetectable if for all probabilistic polynomial-time users  $\mathcal{A}$ , there exists a negligible function  $\epsilon(\lambda)$  such that  $\text{Adv}_{\mathcal{A}, \text{SIG}, \overline{\text{SIG}}}^{\text{detect}}(\lambda) \leq \epsilon(\lambda)$ .

Note that the undetectability of a subverted signature scheme implies that the verification algorithm  $\text{Verify}$  should output the same result regardless of whether a subverted signature or a non-subverted one on the same message is provided as input, i.e.  $\text{Verify}(pk, \bar{\sigma}, m) = \text{Verify}(pk, \sigma, m)$ .

The above definition is related to Bellare et al.’s detection notion for a subverted symmetric encryption scheme [6]. It is also closely related to the similar notion “secret undetectability” presented in Ateniese et al.’s work [2] (which was used in [16]), but the key difference is that in our definition, the generation and presence of the subversion key  $subk$  is more explicit as mentioned earlier in this section.

It should be remarked that while the above undetectability definition provides a good theoretical framework for designing subversion for signature schemes, it does not capture other possible practical attacks such as the timing analysis whereby a user can detect subversion by analyzing the execution time of given signature schemes.

## 4 Our Subverted DSA Scheme

Our subverted DSA scheme exploits the random nonce generation of the original DSA: A nonce of every  $(j + 1)$ -th DSA signature (for  $j \bmod 2 = 0$ ) is generated not by choosing a uniform element but by computing a PRF output taking the subversion key and the first component of the  $j$ -th signature as input. This technique leads to highly efficient subversion.

*Description of the proposed subverted DSA scheme.* First, we describe the original DSA scheme as follows. Let  $\text{DSA} = (\text{Gen}_{\text{DSA}}, \text{Sign}_{\text{DSA}}, \text{Verify}_{\text{DSA}})$  denote the DSA scheme. The key generation algorithm  $\text{Gen}_{\text{DSA}}$  produces  $pk = (\mathbb{G}, q, g, y, \mathbf{G}, \mathbf{H})$  and  $sk = (\mathbb{G}, q, g, x, \mathbf{G}, \mathbf{H})$ , where  $\mathbb{G}$  is a cyclic group of prime order  $q$ , generated by  $g$ ,  $y = g^x$  for the uniform  $x \in \mathbb{Z}_q$ ;  $\mathbf{G} : \{0, 1\}^* \rightarrow \mathbb{Z}_q$  and  $\mathbf{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_q$  are cryptographic hash functions. On input a message  $m$  and a private key  $x$ , the signing algorithm  $\text{Sign}_{\text{DSA}}$  generates a signature  $\sigma = (r, s)$  such that  $r = \mathbf{G}(g^k)$  for random  $k \in \mathbb{Z}_q$  and  $s = k^{-1}(\mathbf{H}(m) + xr) \bmod q$ . On input  $pk = (\mathbb{G}, q, g, y, \mathbf{G}, \mathbf{H})$  of DSA, a message  $m$ , a signature  $\sigma = (r, s)$ ,  $\text{Verify}_{\text{DSA}}$  returns 1 if  $r = \mathbf{G}(g^{\mathbf{H}(m)s^{-1}}y^{rs^{-1}})$  and 0 otherwise.

We now describe our subverted DSA scheme  $\overline{\text{DSA}} = (\overline{\text{Gen}}_{\text{DSA}}, \overline{\text{Sign}}_{\text{DSA}})$  as follows:

- Subversion key generation  $\overline{\text{Gen}}_{\text{DSA}}$ : On input  $1^\ell$ , select a PRF  $F : \{0, 1\}^\ell \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ , pick  $\kappa \in \{0, 1\}^\ell$  uniformly at random and return  $subk = (\text{desc}F, \kappa)$ , where “desc $F$ ” denotes the description of  $F$ .
- Subverted sign  $\overline{\text{Sign}}_{\text{DSA}}$ : On input  $sk = (\mathbb{G}, q, g, x, \mathbf{G}, \mathbf{H})$  of DSA, a message  $m$ , a private signing key  $x$  and a subversion key  $subk$  and state  $\iota = (j, \tau)$ , where  $j$  and  $\tau$  are initialized to 0, perform the following:

If  $j = 0 \bmod 2$

Pick  $k \in \mathbb{Z}_q$  at random and compute  $r = \mathbf{G}(g^k)$  and  $s = k^{-1}(\mathbf{H}(m) + xr) \bmod q$

Set  $\tau = r$  and  $\bar{\sigma} = (r, s)$

Else

Compute  $\tilde{k} = F(\kappa, \tau) \bmod q$  and  $r' = \mathbf{G}(g^{\tilde{k}})$

Set  $\bar{\sigma} = (r', s')$

$j = j + 1$

Set  $\iota = (j, \tau)$

Return  $(\bar{\sigma}, \iota)$

Note that the ASAs on DSA presented in [31] and [32], and the generic subverted signature schemes from [2] all require state to be communicated to the adversary in order to extract the private key. (The state is used to indicate which signature or bit is needed for private key extraction.) In many circumstances, this will not be trivial. In contrast, by collecting *any* three consecutive signatures maximum, the adversary can easily extract of the private key from our subverted DSA.

Moreover, our subverted DSA resists a state reset. The intuition for it is as follows. Whenever a reset query is made,  $j$  is set to 0, which causes the subverted signing algorithm to output a regular DSA signature upon receiving a signing query. Hence a sequence of the subverted signatures always starts with a valid DSA signature in which  $r$  is always an uniform element even after the reset. This is proven formally in the proof of Theorem 2.

Finally we remark that the technique used to construct the above subverted DSA scheme can readily be applied to other discrete-logarithm based signature schemes including the Schnorr signature [24], whose subverted scheme is presented in Appendix .1. It is also possible that the technique can further be generalized to cover all the signature schemes of a similar type but we do not pursue it in this work since the main goal of cryptomalware is to realize efficient ASA rather than to design a generic framework, which often does not result in optimized efficiency of (subverted) cryptographic schemes.

*Key extraction.* We show that the private signing key of the DSA scheme can easily be extracted from utilizing our subverted DSA. Any pair of consecutive subverted signatures that starts with even index  $j$  (i.e.  $j = 0 \pmod 2$ ) can reveal the private signing key  $x$ . Even in the event that  $j$  is odd, by selecting a signature that immediately comes after the failed pair, it can be used to extract the private signing key with probability close to 1. A formal proof is as follows.

**Theorem 1.** *Suppose that three consecutive signatures output by the subverted DSA scheme  $\overline{\text{DSA}}$  are given. Then, the signing key of the DSA signature scheme can be extracted with probability 1.*

*Proof.* Suppose that an adversary  $\mathcal{B}$  obtains three consecutive subverted DSA signatures for three arbitrary messages. Denote the three signatures for message  $m$ ,  $m'$  and  $m''$  by  $\bar{\sigma}_j = (r, s)$ ,  $\bar{\sigma}_{j+1} = (r', s')$  and  $\bar{\sigma}_{j+2} = (r'', s'')$  respectively. Suppose that  $j = 0 \pmod 2$ . Then  $\mathcal{B}$  can recover the secret signing key  $x$  using its subversion key  $\text{subk} = (\kappa, F)$  as follows:

Compute  $\tilde{k} = F(\kappa, r)$ ;  
 Compute  $\bar{x} = \frac{\tilde{k}s' - \text{H}(m')}{r'}$  mod  $q$ ;  
 Return  $\bar{x}$ ;

Note that since

$$g^{\bar{x}} = g^{\frac{\tilde{k}s' - \text{H}(m')}{r'}} = g^{\frac{\text{H}(m') + xr' - \text{H}(m')}{r'}} = g^x = y,$$

the above algorithm succeeds in extracting the private key  $x$ . Hence,  $\mathcal{B}$  extracts the signing key with probability 1.

Now, suppose that  $j \pmod 2 \neq 0$ . Then,  $(j + 1) \pmod 2 = 0$  and hence  $\mathcal{B}$  can use the last two consecutive subverted signatures  $\bar{\sigma}_{j+1} = (r', s')$  and  $\bar{\sigma}_{j+2} = (r'', s'')$  to extract the signing key with probability 1 in exactly the same way it was done with  $\bar{\sigma}_{j+1}$  and  $\bar{\sigma}_{j+2}$ . Thus,  $\text{Adv}_{\mathcal{B}, \overline{\text{DSA}}}^{\text{extract}}(\lambda) = 1$ .  $\square$

*Undetectability.* We show that the subverted DSA scheme is undetectable. As a preliminary, we review the definition of a pseudorandom function (PRF).

**Definition 4.** Let  $F : \{0, 1\}^\ell \times \{0, 1\}^* \rightarrow \{0, 1\}^n$  be a function taking a key  $\mu \in \{0, 1\}^\ell$  and an input  $x \in \{0, 1\}^\rho$ . Let  $\mathcal{A}$  be an adversary with oracle access. Assume that  $\text{Func}_\rho^n$  be a set of all functions from  $\{0, 1\}^\rho$  to  $\{0, 1\}^n$ . Consider the following two games:

**Real** $_{\mathcal{A},F}(\ell)$

1. Pick  $\mu \in \{0, 1\}^\ell$  uniformly at random
2.  $\mathcal{A}^{F(\mu, \cdot)}(\ell)$  outputs  $b \in \{0, 1\}$
3. Return  $b$

**Rand** $_{\mathcal{A},F}(\ell)$

1. Pick  $f \in \text{Func}_\rho^n$  uniformly at random
2.  $\mathcal{A}^{f(\cdot)}(\ell)$  outputs  $b \in \{0, 1\}$
3. Return  $b$

$\mathcal{A}$ 's advantage is defined as  $\text{Adv}_{\mathcal{A},F}^{\text{prf}}(\ell) = |\Pr[\mathbf{Real}_{\mathcal{A},F}(\ell) = 1] - \Pr[\mathbf{Rand}_{\mathcal{A},F}(\ell) = 1]|$ .  $F$  is said to be a pseudorandom function if there exists a negligible function  $\epsilon(\ell)$  such that  $\text{Adv}_{\mathcal{A},F}(\ell) \leq \epsilon(\ell)$ , where the probabilities are taken over the randomness used in  $\mathcal{A}$  and the randomness used in the experiment.

Now we prove the following theorem.

**Theorem 2.** *The detection advantage of the subverted DSA signature scheme  $\overline{\text{DSA}}$  is negligible assuming that the function  $F$  is a PRF.*

*Proof.* The proof proceeds with the following sequence of games.

**Game 0.** This game is the same as the real undetectability game  $\mathbf{Detect}_{\mathcal{A},\text{DSA},\overline{\text{DSA}}}(\lambda)$ . For completeness, we describe it as follows.

Let  $\mathcal{A}$  be an adversary making  $Q_S$  signing queries as described in Definition 3. Suppose that  $\mathcal{A}$  is given  $pk = (\mathcal{G}, p, g, y, \mathbf{G}, \mathbf{H})$  and  $sk = (x, pk)$ .

After setting up the private and public key pair, the game specifies a PRF  $F : \{0, 1\}^\ell \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ , initializes  $j = 0$  and  $\tau = \emptyset$ . The game then picks  $b \in \{0, 1\}$  at random.

If  $b = 0$ , on receiving each signing query  $m$  from  $\mathcal{A}$ , the game responds  $\mathcal{A}$  with a valid DSA signature. On receiving a reset query  $rt$ , set  $j = 0$  and  $\tau = \emptyset$ .

If  $b = 1$ , the game does the following:

If  $\mathcal{A}$  submits a signing query  $m$ ,

If  $j = 0 \bmod 2$

Choose  $k \in \mathbb{Z}_q$  uniformly at random, compute  $r = \mathbf{G}(g^k)$  and  $s = k^{-1}(\mathbf{H}(m) + xr) \bmod q$ , and set  $\tau = r$  and  $\bar{\sigma} = (r, s)$

Else

Compute  $\tilde{k} = F(\kappa, \tau) \bmod q$ , compute  $r' = \mathbf{G}(g^{\tilde{k}})$  and  $s' = \tilde{k}^{-1}(\mathbf{H}(m) + xr') \bmod q$ , and set  $\bar{\sigma} = (r', s')$

$j = j + 1$

Return  $\bar{\sigma}$

If  $\mathcal{A}$  submits a reset query  $rt$ , set  $j = 0$  and  $\tau = \emptyset$

Finally  $\mathcal{A}$  outputs its guess  $b' \in \{0, 1\}$ .

Let  $S_0$  be an event that  $b' = b$ .

From now on, we define  $S_i$  for  $i = 1, 2, \dots$  be events that  $b' = b$ .

**Game 1.** This game modifies the previous game in such a way that when  $b = 0$ , it answers  $\mathcal{A}$ 's queries in the same way as the previous game but when  $b = 1$ ,  $\tilde{k}$  (when  $j = 1 \bmod 2$ ) is not computed

using the PRF but by selecting a uniform random string. As follows is the detailed description of the modified game:

Set  $T = \emptyset$ .

If  $b = 0$ , on receiving each signing query  $m$  from  $\mathcal{A}$ , the game responds  $\mathcal{A}$  with a valid DSA signature. On receiving a reset query  $rt$ , set  $j = 0$  and  $\tau = \emptyset$ .

If  $b = 1$ , the game does the following:

If  $\mathcal{A}$  submits a signing query  $m$ ,

If  $j = 0 \pmod 2$

Choose  $k \in \mathbb{Z}_q$  uniformly at random, compute  $r = G(g^k)$  and  $s = k^{-1}(H(m) + xr) \pmod q$ , and set  $\tau = r$  and  $\bar{\sigma} = (r, s)$

Else

If  $(\tau, t) \notin T$ , pick  $t \in \{0, 1\}^n$  uniformly at random, otherwise extract  $(\tau, t)$  from  $T$ , compute  $\tilde{k} = t \pmod q$ ,  $r' = G(g^{\tilde{k}})$  and  $s' = \tilde{k}^{-1}(H(m) + xr') \pmod q$ , and set  $T = T \cup \{(\tau, t)\}$  and  $\bar{\sigma} = (r', s')$

$j = j + 1$

Return  $\bar{\sigma}$

If  $\mathcal{A}$  submits a reset query  $rt$ , set  $j = 0$  and  $\tau = \emptyset$

Note that other parts of Game 0 remain unchanged.

Observe now that one can construct a PRF adversary  $\mathcal{B}$ , which is able to simulate the subverted signing oracle in Game 0 and Game 1, having access to an oracle  $\mathcal{O} \in \{F(\kappa, \cdot), f(\cdot)\}$ . (When the oracle is the real PRF  $F$ ,  $\tilde{k}$  is produced in the same way as Game 0 is described. On the other hand if the oracle is the random function  $f$ ,  $\tilde{k}$  is produced in the same way as Game 1.) Hence, the difference between the probabilities that the events  $S_0$  and  $S_1$  happen respectively leads to the distinguishing the PRF from the random function. Then, we have

$$|\Pr[S_1] - \Pr[S_0]| = \text{Adv}_{\mathcal{A}, F}^{\text{prf}}(\ell). \quad (1)$$

**Game 2.** This game modifies the subverted signing algorithm in the previous game in a way that when  $j = 1 \pmod 2$ , the existence of  $(\tau, t)$  in  $T$  is not checked and  $t \in \{0, 1\}^n$  is chosen uniformly at random and  $\tilde{k} = t \pmod q$  is computed.

Let  $\text{Col}$  be an event that  $\tau_j = \tau_l$  for some  $l \leq Q_S$ .

Then, we have

$$|\Pr[S_2] - \Pr[S_1]| \leq \Pr[\text{Col}] \leq \frac{Q_S^2}{|q|} \quad (2)$$

since  $\tau$ 's are uniform in  $\mathbb{Z}_q$ .

**Game 3.** This game is identical to the previous game except that when  $\mathcal{A}$  issues a reset query  $rt$ , the game does not reset  $j$  and  $\tau$  (to 0 and  $\emptyset$  respectively) but continues answering  $\mathcal{A}$ 's other queries in the same way it does in Game 2.

Note that in this game, the uniform  $t \in \{0, 1\}^n$  is generated and assigned to  $\tilde{k}$  without depending on the value of  $\tau$ , (i.e.  $\tilde{k}$  is uniform and independent from anything else), the distribution of subverted signatures  $\bar{\sigma}$  is identical to that of the real DSA signatures regardless of whether  $j = 0 \pmod 2$  or  $j = 1 \pmod 2$ . This means that even if the game does not do anything upon receiving the reset query, the distribution of subverted signatures in this game is the same as those in Game 2. Thus, we have

$$\Pr[S_3] = \Pr[S_2]. \quad (3)$$

Note also that in this game, due to the fact that  $\tilde{k}$  is uniform and independent from anything else, regardless of whether  $b = 0$  or  $b = 1$ , upon submitting signing queries,  $\mathcal{A}$  will obtain signatures whose distribution is the same as that of DSA. This means  $\mathcal{A}$  does not get any advantage in guessing  $b$  by querying the signing oracle. Therefore, we have

$$\Pr[S_3] = 1/2. \tag{4}$$

Since  $|\Pr[S_0] - 1/2| = \text{Adv}_{\mathcal{A}, \text{DSA}, \overline{\text{DSA}}}^{\text{detect}}(\lambda)$ , by combining (1), (2), (3) and (4), we obtain

$$\text{Adv}_{\mathcal{A}, \text{DSA}, \overline{\text{DSA}}}^{\text{detect}}(\lambda) \leq \text{Adv}_{\mathcal{B}, F}^{\text{prf}}(\ell) + \frac{Q_S^2}{q},$$

which concludes the proof. □

## 5 Implementation

In this section, we present the implementation of the subverted DSA scheme presented in the previous section. Our implementation is based on the recent version of `Libgcrypt` (v1.8.2), a popular cryptographic library developed for the GNU Privacy Guard (GPG) [11].

Our implementation was performed on Ubuntu 16.04L running on a virtual machine of 2 CPU cores, in which 4096 MB memory was allocated. The CPU of the host PC was Intel(R) Core(TM) i5-7440HQ @ 2.80 GHZ. The OS of the host PC was 64-bit Windows 10 and the memory size was 16 GB. We used primes  $p$  and  $q$ , whose bit-lengths are 1024 and 160, respectively.

As a benchmark, we performed the signing and verification operations of the original (non-subverted) DSA scheme from `Libgcrypt` 200,000 times and measured the running time. The result is provided in Table 1.

Table 1: Execution time of the Original DSA

Algorithm	Exec. Times ( $\mu\text{s}$ )
Average Signing Time	311.7
Average Verification Time	379.5

Algorithm 1 describes the original signing algorithm of the DSA, denoted `DSA_SIGN`. We separated the signing procedure of the original DSA into two functions, `GenkDSA` and `SignDSA`, to easily compare with the subverted signing algorithm, which will be described shortly. `GenkDSA` denotes the generation of randomness  $k$  using one of the random number generators, `_gcry_dsa_gen_rfc6979_k` and `_gcry_dsa_gen_k`, provided by the original `Libgcrypt`. `SignDSA` is the rest of signing process, which mainly consists of simple arithmetic operations.

---

### Algorithm 1 DSA Signing (Not Subverted)

---

- 1: **procedure** `DSA_SIGN`(*Message*, *Sigkey*)
  - 2:    $m \leftarrow \text{Hash}(\textit{Message})$
  - 3:    $k \leftarrow \text{Genk}^{\text{DSA}}(\textit{Sigkey} \rightarrow q)$
  - 4:    $(r, s) \leftarrow \text{Sign}^{\text{DSA}}(\textit{Sigkey}, m, k)$
  - 5:   **return**  $(r, s)$
- 

We then describe our subverted DSA from Section 4 as Algorithm 2.

In Algorithm 2, the static global variables  $\kappa$ ,  $\tau$  and  $j$  represent the subversion key, the variable for saving a previous signature value and the counter, respectively. When  $j\%2 = 0$ , the randomness

---

**Algorithm 2** DSA\_Subversion\_Plain

---

```
1:  $\kappa \leftarrow 0x10234\dots$ 
2:  $\tau \leftarrow 0$ 
3:  $j \leftarrow 0$ 
4: procedure DSA_SUB_SIGN( $Message, Sig_{key}$ )
5:    $k \leftarrow 0$ 
6:    $m \leftarrow \text{Hash}(Message)$ 
7:   if  $j\%2 = 0$  then
8:      $k \leftarrow \text{Genk}^{\text{DSA}}(Sig_{key} \rightarrow q)$ 
9:   else
10:     $k \leftarrow \text{Genk}^{\text{AES256}}(\kappa, \tau, Sig_{key} \rightarrow q)$ 
11:    $(r, s) \leftarrow \text{Sign}^{\text{DSA}}(Sig_{key}, m, k)$ 
12:   if  $j\%2 = 0$  then
13:      $\tau \leftarrow r$ 
14:    $j = j + 1$ 
15:   return  $(r, s)$ 
```

---

variable  $k$  is generated in exactly the same way as done in Algorithm 1, which uses the random number generator provided by `Libgcrypt` for the regular DSA. On the other hand when  $j\%2 = 1$ ,  $k$  is generated by the algorithm  $\text{Genk}^{\text{AES256}}$  based on the pseudorandom function of our choice (AES), taking  $\kappa$  and  $\tau$  as input.

We ran the above Algorithm 2 200,000 times (100,000 times each for the cases  $j\%2 = 0$  and  $j\%2 = 1$ ) and then measured the average signing time for both cases, which is presented in Table 2. Notice that the signing time when  $j\%2 = 1$  is faster than when  $j\%2 = 0$ . This should not be considered as abnormal since the original random generator in `Libgcrypt` requires more complex operations to generate secure randomness than the AES encryption, which we chose to implement the PRF  $F$  in our subverted DSA scheme.

Table 2: Execution times of subverted DSA (Plain)

Algorithm	Exec. Times ( $\mu\text{s}$ )
Average Signing Time When $j\%2 = 0$	313.2
Average Signing Time When $j\%2 = 1$	292.0
Average Signing Time (All Cases)	302.6

## 5.1 Preventing Timing Analysis

The proven undetectability of the subverted DSA (Theorem 2) guarantees that no party can distinguish the real signatures from subverted ones. However, what this theoretical result does not capture is *timing analysis*, whereby anyone tries to detect subversion by comparing the running time of a given algorithm, which is supposed to be subverted, with that of the non-subverted one, which is provided as a benchmark. (Note that there is a similar concept in the literature called “timing attack” [15], which makes it possible for an attacker to measure a time difference between two computational operations in a cryptographic scheme to obtain confidential information such as a secret key.)

The timing analysis might be possible against our implementation of the subverted DSA since in the subversion,  $k$  and  $\tilde{k}$  are generated differently depending on the evaluation of state  $j$ : When  $j\%2 = 0$ ,  $k$  is created by the random number generator provided by `Libgcrypt` (i.e. it is generated by  $\text{Genk}^{\text{DSA}}$ ). Otherwise,  $\tilde{k}$  is generated by the AES function, which sources randomness from the

previous signature value. This difference might lead to a simple detection based on timing analysis. Obviously, the time difference between  $j\%2 = 0$  and  $j\%2 = 1$  is present in Table 2. Although the difference is not significant overall, observing a large number of signatures might lead to detection.

A simple way to prevent the timing analysis is to make all operations in the algorithm take the same amount of time [15]. For example, we can run both  $\text{Genk}^{\text{DSA}}$  and  $\text{Genk}^{\text{AES256}}$  to generate  $k$  and  $\tilde{k}$ , then take one of those values depending on the current state  $j\%2$ . However, this approach will make the signature generation slower since the random values are generated twice, which can lead to detection while the subverted DSA is running. Since our purpose is to make the subversion remain undetectable even at runtime, we cannot take this approach.

Therefore, we suggest new methods to prevent the timing analysis on our subverted DSA, such as inserting dummy operation and resetting the statue index. The rest of this subsection will explain each method in detail.

*Dummy-Operation method.* We were able to reduce the running time difference between the cases  $j\%2 = 0$  and  $j\%2 = 1$  by adding some dummy operation when  $j\%2 = 1$ . (Fortunately because the running in this case is faster, we could easily “slow down” the signature generation.) To achieve this, we carefully measured the extra time required to reduce the time difference and added three extra AES encryptions as a “dummy operation”, denoted by  $Dummy()$  in **Algorithm 3**. Introducing  $Dummy()$  did reduce the difference as shown in Table 3. It also made the overall signature generation time very close to that of the original DSA (cf. Table 1).

Table 3: Execution times of subverted DSA (Dummy Operation)

Algorithm	Exec. Times ( $\mu\text{s}$ )
Average Signing Time When $j\%2 = 0$	314.9
Average Signing Time When $j\%2 = 1$	311.3
Average Signing Time (All Cases)	313.1

---

**Algorithm 3** DSA\_Subversion\_Dummy

---

```

1:  $\kappa \leftarrow 0x10234\dots$ 
2:  $\tau \leftarrow 0$ 
3:  $j \leftarrow 0$ 
4: procedure DSA_SUB_SIGN( $Message, Sig_{key}$ )
5:    $m \leftarrow \text{Hash}(Message)$ 
6:   if  $j\%2 = 0$  then
7:      $k \leftarrow \text{Genk}^{\text{DSA}}(Sig_{key} \rightarrow q)$ 
8:   else
9:      $Dummy()$ 
10:     $k \leftarrow \text{Genk}^{\text{AES256}}(\kappa, \tau, Sig_{key} \rightarrow q)$ 
11:    $(r, s) \leftarrow \text{Sign}^{\text{DSA}}(Sig_{key}, m, k)$ 
12:   if  $j\%2 = 0$  then
13:      $\tau \leftarrow r$ 
14:    $j = j + 1$ 
15:   return  $(r, s)$ 

```

---

*Index-Reset method.* The dummy operation method reduces the time gap between  $j\%2 = 0$  and  $j\%2 = 1$ . In principle, one can “tune” the dummy operation to reduce the time difference further, but we found that it is hard to realize this in practice.

To achieve our goal, we introduce an “index-reset” method, which sometimes resets the index  $j$  to 0. As described in Algorithm 4, every  $\Phi$ -th index is set to 0, where  $\Phi$  is a random integer of a certain size. (As shown in Algorithm 4, the value of  $j$  is set to  $-1$  when  $j\%2 = 0$  to reset the index  $j$ . Since  $j$  is increased by 1 in line 15, it will reset the index  $j$  to 0 at the end of the procedure.) This effectively *skips* the faster signature generation with AES256 when the index is odd (i.e.,  $j\%2 = 1$ ) and replaces it with a slower signature generation with the random number generator provided by the original DSA in `Libgcrypt`.

This idea is based on our observation that when a small number of signatures are generated, the difference between execution time of signature generations when  $j$  is even or odd is not significant, however if the number of generated signatures becomes bigger, say more than 100,000, the difference becomes noticeable. Therefore, by omitting the faster subverted signature generation based on AES, we could make the average signature generation time very much close to that of the original DSA. Note that this does not mean that the subverted signing algorithm omits to sign a particular message, but it rather means that a message is signed by a different subroutine in the algorithm.

---

**Algorithm 4** DSA\_Subversion\_IndexReset

---

```

1:  $\kappa \leftarrow 0x10234\dots$ 
2:  $\tau \leftarrow 0$ 
3:  $j \leftarrow 0$ 
4: procedure DSA_SUB_SIGN( $Message, Sig_{key}$ )
5:    $\Phi \leftarrow N$ 
6:    $m \leftarrow \text{Hash}(Message)$ 
7:   if  $j\%2 = 0$  then
8:      $k \leftarrow \text{Genk}^{\text{DSA}}(Sig_{key} \rightarrow q)$ 
9:      $(r, s) \leftarrow \text{Sign}^{\text{DSA}}(Sig_{key}, m, k)$ 
10:    if  $j\%\Phi = 0$  then
11:       $j = -1$ 
12:    else
13:       $Dummy()$ 
14:       $k \leftarrow \text{Genk}^{\text{AES256}}(\kappa, \tau, Sig_{key} \rightarrow q)$ 
15:       $(r, s) \leftarrow \text{Sign}^{\text{DSA}}(Sig_{key}, m, k)$ 
16:     $j = j + 1$ 
17:  return  $(r, s)$ 

```

---

Table 4 shows the running time of signature generation when both the dummy operation and index-reset methods are applied.

Table 4: Execution times of subverted DSA operation (Index-Reset)

Algorithm	Exec. Times ( $\mu s$ )
Average Generation Time of Every Other Signature Starting with Index = 0	312.3
Average Generation Time of Every Other Signature Starting with Index = 1	311.2
Average Signing Time (All Cases)	311.8

It should be noted that the index-reset method merely affects the performance of the implementation and still allows us to extract a private signing key from consecutive subverted signatures. This

can be shown as follows. If our subverted DSA scheme is reset during the execution, there are only two cases depending on whether  $j\%2 = 0$  or  $j\%2 = 1$  for index  $j$  for the current state. If the algorithm is reset after it generates the signature with index  $j\%2 = 0$ , the next two signatures are generated with indices  $j\%2 = 0$  and  $j\%2 = 1$ , respectively, since we choose because the reset make the signing operation start from  $j = 0$ . (We assume that the resets do not occur immediately one another by choosing  $\Phi$  of a certain size.) Therefore, the sequence of  $j\%2$  is as follows:

$$\dots 0 1 0 1 0 \mathbf{0} 1 \mathbf{0} 1 \dots,$$

where the sequences after the reset are boldfaced. This implies that the longest sequence of  $j\%2$  which is absent from subverted signatures that can lead to the private key extraction is  $1 0 \mathbf{0}$ , where  $\mathbf{0}$  denotes the index that has been reset. If one more signature immediately after “ $\mathbf{0}$ ” is collected, one can extract the private key. Therefore, if maximum four subverted signatures are collected, the private key can be extracted.

If the algorithm is reset after it generates the signature with index  $j\%2 = 1$ , it is obvious that it does not affect the private key extraction as the sequence of  $j\%2$  will be the following:

$$\dots 0 1 0 1 \mathbf{0} 1 \mathbf{0} 1 \dots,$$

which is the same as the original sequence of index for subverted signatures.

To sum up, even when the index-reset is in place, the private key can readily be extracted.

We remark that as proven in Theorem 2, state resets do not affect the undetectability of the subverted DSA. This implies that the index-reset method, which essentially triggers resets on purpose, does not affect the undetectability either.

*Summary of the performance.* Note that the average running time for signature generation of the original DSA is  $311.7 \mu s$  as shown in Table 1. Using the dummy-operation method, we achieved the running time  $313.1 \mu s$  (Table 3). This is slightly slow, but it cannot easily be noticeable and could be regarded as a system noise. The index-reset method gave the better result of running time  $311.8 \mu s$  as shown in Table 4. Note that under this method, the signature generation time for each message does not vary depending on state, as compared with the implementation without timing analysis prevention measure (Algorithm 2). To sum up, our methods effectively prevent the timing analysis giving the signature generation time very close to the benchmark signing time,  $311.7 \mu s$  of the original DSA.

*Comparison with asymmetric subversion.* Compared with our subversion on the DSA, the subversion proposed in [16] could naturally be detectable by timing analysis. Although their subverted signatures are theoretically indistinguishable from the original signatures, the signing algorithm exhibits a noticeable delay because their subverted signing algorithm needs to perform extra exponentiation over a group every time a signature is generated for asymmetric subversion. We estimated that due to the extra exponentiation operation, their subverted signing would take well above  $500 \mu s$  when the `Libgcrypt` is used. This is almost 1.5 times slower than the original signing time of DSA. Therefore, in practice, their subversion can easily be detected by measuring the signature generation time and comparing with a benchmark for the DSA. Our subverted DSA scheme does not have this problem.

## 5.2 Considering Signature Loss

In the next experiment, we consider the real world situation in which the adversary has some difficulty in collecting consecutive signatures. If the adversary can obtain three consecutive signatures, he can definitely recover the private key. However, in practice, he may not be able to get consecutive signatures easily. He can only eavesdrop on a few channels among many, which will make it hard for him to obtain consecutive signatures out of all the distributed signatures. We adopt a probabilistic model to reflect this situation. In our model, we assign a certain probability to each subverted

signature (from the victim’s machine). This probability is represented by a *loss rate*. For example, if the loss rate is 95%, many signatures are marked as “loss” and do not arrive at the adversary’s computer. Then the adversary needs to collect more subverted signatures to successfully extract the private key. Our result shows that in reality, collecting enough signatures (two or three signatures depending on the  $j$  value) is not a very restrictive requirement.

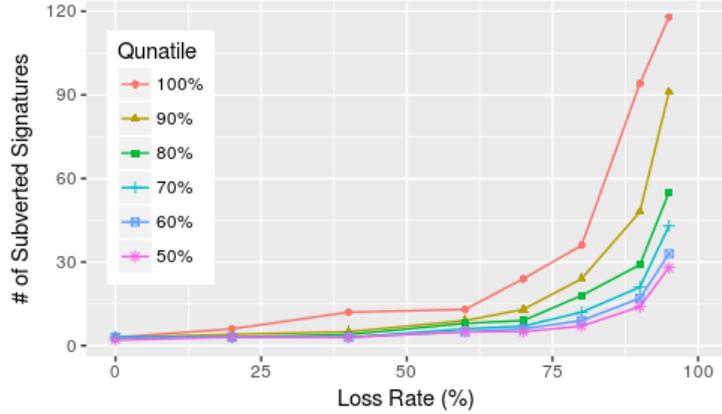


Figure 1: Extraction Results by Quantiles

Fig. 1 shows the relationship between a probability that one can successfully recover a private key and the number of required subverted signatures. It shows the experimental quantiles of the number of signatures required to extract a signing key in each probabilistic scenario. Each curve in the graph represents a probability of successful extractions. The required number of subverted signatures increases as the loss rate increases. For example, if the adversary has gathered about 56 subverted signatures under 95% loss rate, he can recover the private key  $x$  with probability 0.8 (80%), which is represented by the first square dot in the green line in Fig. 1. The number of required signatures to be collected decreases rapidly as the loss rate does as shown in Fig. 1.

Fig. 2 demonstrates the number of subverted signatures that are required on *average* to successfully recover a signing key depending on various loss rates. This result can be used to demonstrate the *expected* resources to extract a private key and it shows that even with high loss rate of 95%, only about 36 signatures are needed for an extracted signing key on average. Consequently, it takes the adversary only about 25.9 ms to work out a signing key. It should be noted that this is computed using average signing, verification and extraction times. For each trial, the adversary signs any message and verifies the signature using a known verification key. If it is verified successfully, then the adversary gets a valid signing key. The time taken to extract the key from two subverted signatures is 50.1  $\mu$ s. The execution time will decrease rapidly as the adversary gathers signatures more reliably without big losses (i.e., the loss rate decreases).

To summarize, our experiment demonstrates that requiring at most three consecutive signatures does not limit the practicality of our subverted DSA scheme. Even in the environment where many subverted signatures are lost in transmission, the adversary can still extract private signing keys utilizing a relatively small number of collected signatures.

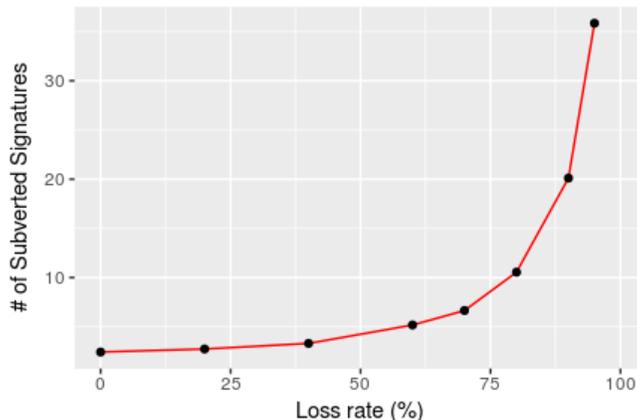


Figure 2: Overall Extraction Results

## 6 Concluding Remarks

We have presented our subverted DSA scheme which realizes ASA on the DSA and provided its implementation based on the `Libgcrypto` library. We have demonstrated that subverted signatures generated from the subverted DSA scheme can be utilized to recover the private signing key from very few signatures. Our experiment shows that even if many subverted signatures are lost en route to the adversary, the private key can still be extracted efficiently.

As with the case of other ASAs in the literature, a lesson learned from our result is that care must be taken to employ black box cryptography. Our result also confirms that randomized cryptographic schemes can be more vulnerable to subversion attacks since the randomness used in such schemes can be a vehicle for Big Brother to embed a backdoor, as demonstrated in a number of subversion attacks in the literature. Moreover, our result demonstrates that the structures of the discrete-logarithm based signatures are particularly vulnerable to exploitation. (Note that the structural fragility of DSA was well demonstrated by the attack on the DSA implementation in the Sony PlayStation3, in which a new ephemeral exponent, say  $k$ , failed to be uniformly random [7].) We expect that other similar discrete-logarithm based signatures can be subverted in a similar way as presented in this paper.

As discussed in Section 2, the research community has come up with a few solutions to defend cryptography against ASA, which include using deterministic cryptographic primitives and offline components. Our concern, however, is that solutions based on ASA framework may not cover all kind of the attempts that weaken cryptographic protocols as elaborated by Schneier et al. [23]. Another concern is that many cryptographic algorithms (such as RSA, ElGamal and DSA) are widely deployed and it will be hard to persuade users to replace what they are currently using, in a short period of time.

As a short-term solution, we then suggest that a proper self-test routine to detect any unauthorized changes in cryptographic implementations should be added. Also, it is important to remind the users of cryptographic libraries 1) to use the versions of software which are legitimately distributed and 2) to check the integrity of sources or binaries proactively before using them.

In the long term, we envision that such a discipline as “software engineering for cryptography”, which was put forward by Schneier et al. [23], is urgently needed to extend the efforts on providing implementation security [1][8] and finding bugs or problems [14] to preserve the security and privacy that cryptography can bring us.

## References

- [1] J. Almeida, M. Barbosa, G. Barther and F. Dupressoir, *Certified Computer-Aided Cryptography: Efficient Provably Secure Machine Code from High-Level Implementations*, In ACM-CCS '13, pp. 1217–1230, 2013.
- [2] G. Ateniese, B. Magri and V. Venturi, *Subversion-Resilient Signature Schemes*, Cryptology ePrint Archive, 2015/517.
- [3] M. Bellare and V. T. Hoang, *Resisting Randomness Subversion: Fast Deterministic and Hedged Public-key Encryption in the Standard Model*, In Eurocrypt '15, LNCS 9057, pp. 627–656, Springer, 2015.
- [4] M. Bellare, J. Jaeger and D. Kane, *Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks*, In ACM-CCS '15, pp. 1431–1440, 2015.
- [5] M. Bellare and P. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, In ACM-CCS '93, pp. 62–73, 1993.
- [6] M. Bellare, K. G. Paterson and P. Rogaway, *Security of Symmetric Encryption against Mass Surveillance*, In Crypto '14, LNCS 8616, pp. 1–19, Springer, 2014.
- [7] M. Bendel, *Hackers Describe PS3 Security As Epic Fail, Gain Unrestricted Access*, Exophase.com. Available at <http://www.exophase.com/20540/hackers-describe-ps3-security-as-epic-fail-gain-unrestricted-access/>
- [8] B. Blanchet, *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*, In IEEE Computer Security Foundations Workshop '01, pp. 82–96, 2001.
- [9] J. Buchmann, *The Digital Signature Algorithm (DSA)*, Cryptography Research and Evaluation Committees (CRYPTREC), Technical Report, No. 1003, 2001. Available at [http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1003\\_DSA.pdf](http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1003_DSA.pdf)
- [10] FIPS PUB 186: Digital Signature Standard (DSS), 1994-05-19. (Four revisions have been released: FIPS 186-1 in 1996, FIPS 186-2 in 2000, FIPS 186-3 in 2009 and FIPS 186-4 in 2013.)
- [11] GnuPG, *The GNU Privacy Guard*, <https://www.gnupg.org/>
- [12] GnuPG, *LIBGCRYPT*, <https://www.gnupg.org/software/libgcrypt/index.html>
- [13] The Guardian, *Revealed: how US and UK spy agencies defeat internet privacy and security*, September 2013, Available at <https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security>
- [14] L. Huang, A. Rice, E. Ellingsen and C. Jackson, *Analyzing Forged SSL Certificates in the Wild*, In IEEE Symposium on Security and Privacy '14, pp. 83–97, 2014.
- [15] P. C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, In CRYPTO '96, LNCS 1109, pp 104–113, Springer, 1996.
- [16] C. Liu, R. Chen, Y. Wang and Y. Wang, *Asymmetric Subversion Attacks on Signature Schemes*, In ACISP '18, LNCS 10946, pp. 376–395, Springer, 2018.
- [17] I. Mironov and N. Stephens-Davidowitz, *Cryptographic Reverse Firewalls*, In Eurocrypt '15, LNCS 9057, pp. 657–686, Springer, 2015.

- [18] National Institute of Standards and Technology, *Recommendations for Random Number Generation Using Deterministic Random Bit Generators (Revised)*, January 2012. NIST SP 800-90.
- [19] P. Nguyen and I. Shparlinski, *The Insecurity of the Digital Signature Algorithm with Partially Known Nonces* J. Cryptology 15(3): pp. 151–176, 2002, Springer.
- [20] D. Pointcheval and J. Stern, *Security Arguments for Digital Signatures and Blind Signatures* J. Cryptology 13(3): pp. 361–396, 2000, Springer.
- [21] A. Russell, Q. Tang, M. Yung and H. Zhou, *Generic Semantic Security against a Kleptographic Adversary*, In ACM-CCS '17, pp. 907–922, 2017.
- [22] B. Schneier, *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*, ISBN-13: 978-0393244816, W. W. Norton & Company, 2015.
- [23] B. Schneier, M. Fredrikson, T. Kohno and T. Ristenpart, *Surreptitiously Weakening Cryptographic Systems*, Cryptology ePrint Archive, 2015/097.
- [24] C. P. Schnorr, *Efficient identification and signatures for Smart Cards*, In Crypto '89, LNCS 435, pp. 239–252, Springer-Verlag, 1990.
- [25] G. J. Simmons, *The Subliminal Channel and Digital Signatures*, In Eurocrypt '84, LNCS 209, pp. 364–378, Springer-Verlag, 1984.
- [26] G. J. Simmons, *Subliminal Communication Is Easy Using the DSA*, In Eurocrypt '93, LNCS 765, pp. 218–232, Springer-Verlag, 1993.
- [27] US-CERT, Alert (TA13-309A) *CryptoLocker Ransomware Infections*, November 05, 2013.
- [28] US-CERT, Alert (TA17-132A) *Indicators Associated With WannaCry Ransomware*, May 12, 2017.
- [29] A. Young and M. Yung, *The Dark Side of “Black-Box” Cryptography, or: Should We Trust Capstone?*, In Crypto '96, LNCS 1109, pp. 89–103, Springer, 1996.
- [30] A. Young and M. Yung, *Kleptography: Using Cryptography Against Cryptography*, In Eurocrypt '97, LNCS 1233, pp. 62–73, Springer, 1997.
- [31] A. Young and M. Yung, *The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems*, In Crypto '97, LNCS 1294, pp. 264–276, Springer, 1997.
- [32] A. Young and M. Yung, *Malicious Cryptography: Exposing Cryptovirology*, ISBN: 0-7645-4975-8, Wiley Publishing, 2004.

## .1 Our Subverted Schnorr Signature

As an additional application of our technique, we present an ASA on Schnorr by constructing a subverted Schnorr scheme. The main idea of the construction is similar to the subverted DSA scheme: A nonce of every  $(j + 1)$ -th DSA signature (for  $j \bmod 2 = 0$ ) is generated by computing a PRF output taking the subversion key and the first component of the  $j$ -th signature (generated legitimately) as input. A formal description follows.

Let  $\text{SNR} = (\text{Gen}_{\text{SNR}}, \text{Sign}_{\text{SNR}}, \text{Verify}_{\text{SNR}})$  denote the Schnorr signature scheme. The key generation algorithm  $\text{Gen}_{\text{SNR}}$  produces  $pk = (\mathbb{G}, q, g, y, \mathbf{H})$  and  $sk = (\mathbb{G}, q, g, x, \mathbf{H})$ , where  $\mathbb{G}$  is a cyclic group of prime order  $q$ , generated by  $g$ ,  $y = g^x$  for uniform random  $x \in \mathbb{Z}_q$  and  $\mathbf{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_q$  is a cryptographic hash function. (This is assumed as a random oracle [5] in the security analysis [20].)

On input a message  $m$  and a private key  $x$ , the signing algorithm  $\text{Sign}_{\text{SNR}}$  generates a signature  $\sigma = (r, s)$  such that  $e = \text{H}(m, r)$ , where  $r = g^k$  for uniform  $k \in \mathbb{Z}_q$ , and  $s = k + xe \pmod q$ . On input a message  $m$ , a signature  $\sigma = (r, s)$  and a public key  $pk$ , the verification algorithm  $\text{Verify}_{\text{SNR}}$  computes  $g^s y^{-e}$  and checks whether  $e = \text{H}(m, g^s y^{-e})$ .

Using this scheme, we can construct a subverted Schnorr signature scheme  $\overline{\text{SNR}} = (\overline{\text{Gen}}_{\text{SNR}}, \overline{\text{Sign}}_{\text{SNR}}, \overline{\text{Verify}}_{\text{SNR}})$  as follows:

- Subversion key generation  $\overline{\text{Gen}}_{\text{SNR}}$ : On input  $1^\ell$ , select a PRF  $F : \{0, 1\}^\ell \times \mathbb{G} \rightarrow \mathbb{Z}_q$ , pick  $\kappa \in \{0, 1\}^\ell$  uniformly at random and return  $\text{sub}k = (\text{desc}F, \kappa)$ , where “desc $F$ ” denotes the description of  $F$ .
- Subverted sign  $\overline{\text{Sign}}_{\text{SNR}}$ : On input  $sk = (\mathbb{G}, q, g, x, \text{H})$  of  $\text{SNR}$ , a message  $m$ , a private signing key  $x$  and a subversion key  $\text{sub}k = (F, \kappa)$  and state  $\iota = (j, \tau)$ , where  $j$  and  $\tau$  are initialized to 0, perform the following:

If  $j = 0 \pmod 2$

Pick  $k \in \mathbb{Z}_q$  uniformly at random and compute  $r = g^k, e = \text{H}(m, r)$  and  $s = (k + xe) \pmod q$

Set  $\tau = r$  and  $\bar{\sigma} = (r, s)$

Else

Compute  $\tilde{k} = F(\kappa, \tau), r' = g^{\tilde{k}}, e' = \text{H}(m, r')$  and  $s' = (\tilde{k} + xe') \pmod q$

Set  $\bar{\sigma} = (r', s')$

$j = j + 1$

Set  $\iota = (j, \tau)$

Return  $(\bar{\sigma}, \iota)$

Note that the above scheme maintains state  $j$  (meaning it is stateful) but the subverted signing algorithm never outputs the state nor does it take the state as input. – It is only used internally. Unlike the subverted signature schemes in the literature [2], [31], [30], the state need not be communicated to the adversary to conduct further attacks like key extraction.

Note also that the subverted verification algorithm is exactly the same as the original verification algorithm of the Schnorr signature scheme, which is a required property to prove undetectability in the next subsection.

*Key extraction.* The key extraction process is highly efficient in that *any* pair of consecutive subverted signatures, which starts with even index  $j$  can reveal the private signing key  $x$ . (Recall that consecutive signatures are the signatures generated one after another.) If  $j$  is an odd number, i.e.  $j = 1 \pmod 2$ , extraction will fail. However, this can easily be overcome by sampling *three* consecutive subverted signatures so that if the first two signatures fail to extract the signing key and last pair can be chosen to extract it successfully. The following theorem formally proves the above argument.

**Theorem 3.** *Suppose that three consecutive signatures output by the subverted Schnorr signature scheme  $\overline{\text{SNR}}$  are given. Then, the signing key of the SNR signature scheme can be extracted with probability 1.*

*Proof.* Suppose that an adversary  $\mathcal{B}$  obtains three consecutive subverted Schnorr signatures for three messages. Denote the three signatures for message  $m, m'$  and  $m''$  by  $\bar{\sigma}_j = (r, s), \bar{\sigma}_{j+1} = (r', s')$  and  $\bar{\sigma}_{j+2} = (r'', s'')$  respectively. Suppose that  $j = 0 \pmod 2$ . Then  $\mathcal{B}$  can recover the secret signing key  $x$  as follows:

Compute  $e = \text{H}(m, r)$  and  $e' = \text{H}(m, r')$ ;

Compute  $\tilde{k} = F(\kappa, g^s y^{-e})$ ;

Compute  $\bar{x} = \frac{s' - \tilde{k}}{e'} \bmod q$ ;

Return  $\bar{x}$ ;

Since

$$g^{\bar{x}} = g^{\frac{s' - \tilde{k}}{e'}} = g^{\frac{\tilde{k} + x e' - \tilde{k}}{e'}} = g^x = y$$

by the subverted signing algorithm, the above algorithm succeeds in extracting the private key  $x$ . Hence,  $\mathcal{B}$  successfully extracts the signing key with probability 1.

When  $j \neq 0 \bmod 2$ ,  $\mathcal{B}$  can use  $\bar{\sigma}_{j+1}$  and  $\bar{\sigma}_{j+2}$  to extract the signing key in a similar way as shown in the proof of Theorem 1. Hence,  $\text{Adv}_{\mathcal{B}, \overline{\text{SNR}}}^{\text{extract}}(\lambda) = 1$ .  $\square$

*Undetectability.* We were also able to show that the proposed subverted Schnorr signature scheme is undetectable as stated in the following theorem.

**Theorem 4.** *The detection advantage of the subverted Schnorr signature scheme  $\overline{\text{SNR}}$  is negligible assuming that the function  $F$  is a PRF.*

The proof is very similar to that of Theorem 2, which will be provided in the full version of the paper.