On Access Control Encryption Without Sanitization

Cecilia Boschini[1](✉), Ivan Damgård[2], and Claudio Orlandi[2]

[1] Technion and Reichman University (IDC), Haifa, Herzliya, Israel
cecilia.bo@cs.technion.ac.il
[2] Aarhus University, Aarhus, Denmark
{ivan,orlandi}@cs.au.dk

Abstract. Access Control Encryption (ACE) [4] allows to control information flow between parties by enforcing a policy that specifies which user can send messages to whom. The core of the scheme is a sanitizer, i.e., an entity that "sanitizes" all messages by essentially re-encrypting the ciphertexts under its key. In this work we investigate the natural question of whether it is still possible to achieve some meaningful security properties in scenarios when such a sanitization step is not possible. We answer positively by showing that it is possible to limit corrupted users to communicate only through insecure subliminal channels, under the necessary assumption that parties do not have pre-shared randomness. Moreover, we show that the bandwidth of such channels can be limited to be $O(\log(\lambda))$ by adding public ciphertext verifiability to the scheme under computational assumptions. In particular, we rely on a new security definition for obfuscation, Game Specific Obfuscation (GSO), which is a weaker definition than VBB, as it only requires the obfuscator to obfuscate programs in a specific family of programs, and limited to a fixed security game.

1 Introduction

Designers of practical secure IT systems are often interested in controlling the flow of information in their system. For this purpose one sets up a security policy that contains rules on what operations the entities in the system are allowed to execute. Crucially, such rules must constrain both write and read operations as both types may lead to unwanted transfer of data. This was formalized in the classical Bell-Lapadula security policy as the "no read up" (entities with low security clearance cannot read top-secret data) and "no write-down" rules (entities with a high security clearance cannot write to public files). If entities are not assumed to be honest, such a security policy cannot be enforced unless we assume a trusted party, often known as a *sanitizer*, which will stop and/or modify unwanted communication. Of course, the sanitizer cannot do this unless we assume that parties can only communicate via the sanitizer. In practical systems one usually tries to ensure this by a combination of hardware security and software design, for instance in the kernel of the operating system.

https://doi.org/10.1007/978-3-031-14791-3_10

In [4] Damgård et al. asked whether cryptography can be used to simplify the job of the sanitizer, and reduce the amount of trust we need to place in it. To this end, they introduced the notion of Access Control Encryption (ACE). Using an ACE scheme, the sanitizer does not need to know the security policy or the identities of any parties in the system. It just needs to process every message it receives and pass it on. The processing essentially amounts to re-randomize every message received. Instead of asking the sanitizer to enforce the security policy, an ACE scheme integrates the policy in the key generation algorithm, which gives an encryption key to each sender, a decryption key to each receiver and a sanitizer key for the sanitizer. The keys are designed such that, after sanitization, a receiver can decrypt a message, only if it was encrypted by a sender that is allowed to send to that receiver.

Observe that security requires the physical assumption that a corrupt sender cannot bypass the sanitizer and send directly to any receiver she wants. Indeed, it may seem that nothing non-trivial can be achieved if we drop this assumption. On the other hand, assuming such a communication bottleneck may be hard to justify in practice, and makes the system vulnerable to DDoS attacks (in case the sanitizer is offline). It is then natural to wonder:

Can we achieve any meaningful security without sanitization?

2 Our Results

In this paper, we answer affirmatively to the previous question analyzing two new models, both avoiding the need of preprocessing ciphertexts before delivery. We present formal definitions of ACE in these models, and we instantiate them under various computational assumptions. Along the way we obtain a standard ACE with sender anonymity from standard assumptions, which had been left as an open problem in [6] (deferred to the full version due to lack of space).

2.1 Modeling ACE Without Sanitization

ACE without Sanitizer (ACEnoS). Removing the sanitization bottleneck implies that senders can now post to a bulletin board that receivers can read from. As in standard ACE, parties have no other communication channel available, and key generation assumes a trusted party. However, senders are now free to post whatever they want. What security properties can we hope to achieve in such a model? Clearly, we can do what cryptography "natively" allows us to do, namely what we call the *No Read Rule* (NRR): an honest sender can encrypt a message such that only the designated receiver can extract information about the plaintext from the ciphertext; furthermore we can guarantee that a ciphertext does not reveal the identity of the sender. What we can do about a corrupt sender is more subtle: clearly, we cannot stop a sender from simply posting any message she wants, thus broadcasting confidential data. But, on the other hand, this is often not what a corrupt sender wants to do. If, for instance, the data involved is extremely valuable, it may be more attractive to break the security policy by sending a *secret* message that can only be decrypted by a specific (corrupted) receiver she is not allowed to send to. This attack we can actually hope to stop, through what we call the *No Secret Write Rule* (NSWR)[1]: parties cannot communicate secretly if the policy does not allow it. If parties manage to communicate against policy, then anyone can read their communication.

Communication Restrictions. For this goal to be meaningful, we can allow corrupted senders and receivers to share a common strategy, but not randomness. Without this constraint, any no secret write rule can trivially be broken just using one-time-pad encryption, for instance. Assuming that the parties' initial states are uncorrelated, the rough intuition is that if the key generation does not supply a corrupted sender and receiver with sufficiently correlated key material, the receiver's ability to decrypt a ciphertext cannot depend on the keys she has. But if it does not, then anyone should be able to extract the message the corrupted sender wants to leak, and so the message is effectively publicly available. Observe that the assumption that parties do not have pre-shared randomness is not new: in fact, Alwën et al. [1] already pointed out the need for such an assumption when building collusion-free protocols.

Verifiable ACE (VACE). Our solution above implies that whatever information a corrupt sender embeds in his message can in principle be accessed by anyone. But there is no limit on the *amount* of information she can leak in this way. Is there some way to plausibly limit such leakage? We answer affirmatively, by adding a way to publicly verify the posted ciphertexts. Intuitively, if a ciphertext verifies, it is correctly formed according to the encryption algorithm, not something the sender can choose as he likes (e.g., no unencrypted messages). However, a sender may still try to output a valid ciphertext that equals the encoding of an n-bit subliminal message. The hope is that now the sender's situation becomes somewhat similar to having to generate ciphertexts by calling a random (encryption) oracle. In this scenario embedding a random n-bits string requires a number of queries exponential in n, as the sender can only make a polynomial number of calls and cannot control the (somewhat) random outputs. This limits senders to leak up to a logarithmic number of bits, which is optimal[2].

Finally, as anyone can verify, senders are discouraged from posting invalid ciphertexts (e.g., unencrypted messages) – as in practice, content that does not verify would be taken down and there might be consequences for the sender. With this we obtain fast communication (no need of a sanitization bottleneck), while maintaining some accountability. Observe that public verification yields something different from a standard ACE, albeit very close. The difference is

[1] This is closely related to the notion of subliminal channels [10], where the information sent is hidden in messages that are seemingly created for a different purpose. In that language, NSWR says that, while a corrupt sender may be able to establish a subliminal channel to a receiver he should not send to, any such channel is non-secret.

[2] This reasoning yields a clear lower bound: no ACEnoS can prevent a sender to embed a logarithmic number of bits in a ciphertext (by generating ciphertexts until, say, the first few bits of the string are equal to the message bits she wants to embed).

that not only the sanitization key is public (as in [5]), but the sanitization step (the verification in this case) can be performed by *any* party, after ciphertexts are posted. This was not the case in [5], where the sanitizer does more than just a routine check (in fact, it injects honestly generated randomness in ciphertexts).

2.2 Instantiating ACEnoS and VACE

Constructing ACEnoS. Even assuming parties not to have shared randomness, it is not straightforward to obtain a ACEnoS by simply "removing" the sanitization step from pre-existing ACE constructions: the security of existing ACE schemes strongly relies on some transformation to be applied on a ciphertext *before* its delivery. In our work, we give several constructions under various standard assumptions that match in efficiency the existing ACE constructions (e.g., [4,5]). One of these requires a new primitive, key-indistinguishable predicate encryption. The definition is rather natural, and very useful, as it immediately yields a solution of a problem left open by Kim and Wu [6] (see full version).

Constructing VACE. We give a construction of an ACE scheme with verification and minimal leakage based on a new definition of obfuscation. The need of a new assumption arises from the fact that building a VACE is highly non-trivial. To see why, we can consider what seems at first a promising solution: assume the sender is committed to a PRF key K and is supposed to compute the ciphertext c she posts using randomness generated from K and the encrypted message m, via the PRF, i.e., $c = E_{pk}(m, \mathsf{PRF}_K(m))$. In addition, the sender adds a non-interactive zero-knowledge proof that c was correctly computed. This allows verification. Moreover, it also seems to imply that a malicious sender cannot manipulate the randomness to embed a subliminal message m' in c. However a closer look shows that this is not clear at all: the intuition assumes that the sender chooses a message m to encrypt and the subliminal message m' first, then generates randomness using the PRF key and hopes that the resulting ciphertext will be an encoding m'. In fact, the sender does not have to do this: she might be able to instead compute simultaneously c and m from the subliminal message m', in a way that depends on K, such that $c = E_{pk}(m, \mathsf{PRF}_K(m))$ holds. The security properties of the PRF and the encryption function do not imply that this is infeasible: the PRF is only secure if the key is not known, and the encryption function is only hard to invert on a random ciphertext, and this does not prevent the adversary from generating c and m simultaneously from K. With this approach, it is completely unclear that c could not be an encoding of a subliminal message m' that the adversary wants. One might be able to make these problems go away if one is willing to model the PRF as a random oracle. But now the problem is that the zero-knowledge proof requires access to the code of the instantiation of the oracle. This code is no longer available once we pass to the random oracle model, so it is not clear how to prove security.

In the absence of a solution based on standard assumptions, we rely on a new model for security of obfuscation that we call Game-Specific Obfuscation (GSO). As the name suggests, GSO only requires the obfuscator to obfuscate programs in a specific family of programs $\mathcal{F}$ used in a fixed security game G. Roughly speaking, the security requirement is that the obfuscated program does not help an adversary to win the specific game any more than oracle access to the program would have allowed. Note that while implied by VBB, GSO makes a much weaker demand than VBB: we assume that the obfuscation gives nothing more than oracle access, only as far as winning G is concerned, and the obfuscator only needs to obfuscate programs in $\mathcal{F}$. In particular, the impossibility result for VBB [2] does not apply to GSO. At the same time, GSO and iO are somewhat incomparable: GSO has no specific requirement on the family of programs, while iO needs them to compute the same function; on the other hand, iO still guarantees indistinguishability for every game, while GSO targets a specific one. Nevertheless, assuming GSO is a strong assumption, and our result mainly serves to rule out impossibility results for VACE with minimal leakage.

2.3 Concurrent Work

Recently, Lu et al. [7] explore an analogous question in the context of collusion-preserving MPC [1]: could one get rid of mediation? At a high level, their solution is similar to our VACE construction: parties' messages are encrypted, signed, and sent on an authenticated broadcast channel by a trusted hardware, which thus performs the same task as the obfuscated program in our construction. However, to completely prevent subliminal channels they have to assume that senders cannot run the trusted hardware multiple times and choose which ciphertext to send, which is a stronger assumption than our communication model.

2.4 Future Directions

We believe that the question we study here is a fundamental one that is of interest, also outside the scope of ACE, as it can be phrased in a much more general context: assume a polynomial-time sender who is limited to sending messages that satisfy some verification predicate. The question is to what extent we can use the verification to limit the bandwidth of any subliminal channel that the sender may be able to embed? Given our results, it seems that a logarithmic number of bits per message can be achieved. However, we leave a solution based on standard assumptions as an open problem.

3 Access Control Encryption Without Sanitization

Let $[n] = 0, 1, \ldots, n$ for an integer $n \in \mathbb{N}$, and λ be the security parameter. Denote by $|s|$ the length of a bit string s.

Parties. The protocol is run by n parties P_i. Each party can be *either* a sender *or* a receiver. We denote by n_S (resp., n_R) the number of senders (resp., receivers); thus $n = n_S + n_R$.

Policy. A policy is a function $\mathcal{P} : [n_S] \times [n_R] \to \{0, 1\}$ defined as follows:

- $\mathcal{P}(i,j) = 1$ means that the i-th sender can send messages to the j-th receiver (i.e., R_j can decrypt ciphertexts generated by S_i);
- $\mathcal{P}(i,j) = 0$ means that the i-th sender cannot send messages to the j-th receiver (i.e., R_j cannot decrypt ciphertexts generated by S_i);

Finally, the party identity $i = 0$ represents a sender or receiver with no rights, i.e., for all $j \in [n_R]$, $k \in [n_S]$ it holds $\mathcal{P}(0,j) = \mathcal{P}(k,0) = 0$.

Communication Model. We assume only one-way channels between parties:

- parties cannot share any randomness nor other key setup, and
- parties only communicate through a bulletin board, and do not have private channels, or, in general, communication channels outside the protocol (analogously to ACE). Senders are the only ones allowed to write on the bulletin board, while receivers have read-only access to it.

An Access Control Encryption scheme without sanitizer (denoted by ACEnoS in this work) is composed by four algorithms:

Setup: $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$
Takes as input the security parameter λ and the policy $\mathcal{P}$, and outputs the public parameters of the scheme (that include the message space $\mathcal{M}$) and the master secret key.

Key Generation: $k_i \leftarrow \mathsf{KGen}(pp, msk, i, t)$
Takes as input the public parameters of the scheme, the master secret key, the identity of the party, and a type $t \in \{\mathsf{sen}, \mathsf{rec}\}$, and outputs a key k_i, generated depending on t and i as follows:

- $ek_i \leftarrow \mathsf{KGen}(pp, msk, i, \mathsf{sen})$ is the encryption key for $i \in [n_S]$;
- $dk_i \leftarrow \mathsf{KGen}(pp, msk, i, \mathsf{rec})$ is the decryption key for $i \in [n_R]$;
- $ek_0 = dk_0 = pp$.

Encryption: $c \leftarrow \mathsf{Enc}(pp, ek_i, m)$
On input the secret key of S_i and a message $m \in \mathcal{M}$, outputs the ciphertext.

Decryption: $m' \leftarrow \mathsf{Dec}(pp, dk_i, c)$
On input a ciphertext and the secret key of the receiver i, it outputs either a message or $\perp$ (representing a decryption failure).

As in the original scheme, an ACE without sanitizer has to satisfy:

Correctness: a honestly generated ciphertext can always be decrypted by the designated receivers.

No Read Rule: only the designated receiver can extract information about the plaintext from a ciphertext; senders anonymity is guaranteed under natural assumptions.

No *Secret* Write Rule: parties cannot communicate secretly if the policy does not allow it. If parties manage to communicate despite being forbidden by the policy, then anyone can read their communication.

When compared to the security definitions of ACE, only the No write Rule requires major changes, as it is the only property where the sanitizer plays a fundamental role. Correctness and the No Real Rule only need small adjustments.

Definition 3.1 (Correctness). *An ACE without sanitizer is correct if for all $m \in \mathcal{M}$, $i \in [n_S]$, $j \in [n_R]$ such that $\mathcal{P}(i,j) = 1$ it holds*

$$\Pr\left[\mathsf{Dec}(pp, dk_j, \mathsf{Enc}(pp, ek_i, m)) \neq m \;:\; \begin{array}{r} (pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P}), \\ ek_i \leftarrow \mathsf{KGen}(pp, msk, i, \mathsf{sen}), \\ dk_j \leftarrow \mathsf{KGen}(pp, msk, j, \mathsf{rec}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probabilities are taken over the random coins of all the algorithms.

The NRR models the case in which a coalition of parties (both senders and receivers) tries to either break the confidentiality of a message (payload privacy) or to break the anonymity of target senders. We consider the most powerful adversary, that has even access to the target senders' encryption keys. This guarantees sender's anonymity (and payload privacy) even for senders whose encryption key was leaked.

Definition 3.2 (No-Read Rule). *Consider the following security experiment, where* A *is a stateful adversary and* $b \in \{0,1\}$,

Experiment $\mathsf{Exp}^{\mathsf{nr}}_{\mathsf{A},b}(\lambda, \mathcal{P})$	*Oracles*	
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$	$\mathcal{O}_G(j,t)$:	$\mathcal{O}_E(j,m)$:
$(m_0, m_1, i_0, i_1, st) \leftarrow \mathsf{A}^{\mathcal{O}_G(\cdot),\ \mathcal{O}_E(\cdot)}(pp)$	If $\exists\ k_j$ s.t. $(k_j, j, t) \in \mathcal{L}$, return k_j	$ek_j \leftarrow \mathcal{O}_G(j, \mathsf{sen})$
$c_b \leftarrow \mathsf{Enc}(pp, \mathcal{O}_G(i_b, \mathsf{sen}), m_b)$	*Else* $k_j \leftarrow \mathsf{KGen}(pp, msk, j, t)$	$c \leftarrow \mathsf{Enc}(pp, ek_j, m)$
$b' \leftarrow \mathsf{A}^{\mathcal{O}_G(\cdot),\ \mathcal{O}_E(\cdot)}(st, c_b)$	$\mathcal{L} \leftarrow \mathcal{L} \cup \{(k_j, j, t)\}$	*Return* c.
Return b'.	*Return* k_j.	

Given the following requirement,

Necessary Condition: $b = b'$, $|m_0| = |m_1|$, $i_0, i_1 \in [n_S]$,

we say that A *wins the experiment if one of the following holds:*

Payload Privacy *(*PP*). The Necessary Condition holds, and for all queries* $q = (j, \mathsf{rec})$ *to* $\mathcal{O}_G$ *it holds that:* $\mathcal{P}(i_0, j) = \mathcal{P}(i_1, j) = 0$.

Sender Anonymity *(*SA*). The Necessary Condition holds, and for all queries* $q = (j, \mathsf{rec})$ *to* $\mathcal{O}_G$ *it holds that:* $\mathcal{P}(i_0, j) = \mathcal{P}(i_1, j)$ *and* $m_0 = m_1$.

An ACE without sanitizer satisfies the No-Read rule if for all PPT A, $b \xleftarrow{\$} \{0,1\}$

$$2 \cdot \left| \Pr\left[(\mathsf{PP} \ \vee\ \mathsf{SA}) \;:\; b' \leftarrow \mathsf{Exp}^{\mathsf{nr}}_{\mathsf{A},b}(\lambda, \mathcal{P})\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

NRR vs. Indistinguishability. The NRR corresponds to the indistinguishability properties of the PKE, which in fact can be seen as special cases of the NRR: Payload Privacy when $i_0 = i_1$ guarantees IND-CPA security, while the Sender Anonymity case is analogous of key-indistinguishability [3].

The goal of the No Secret Write Rule is to prevent unauthorized communications. However, as the sanitization step is not present anymore, there is no countermeasure in place to prevent parties to try to establish subliminal channels [10]: parties might try to embed messages in the bits of a valid ciphertext using some shared randomness (for example, bits of their secret keys). As completely preventing subliminal channels without some kind of sanitization step is impossible (cf. Sect. 7.2), we settle for preventing *secure* exfiltration of information: if two parties manage to communicate despite this being against the policy, they can only succeed in establishing an insecure subliminal channel (i.e., they can only send unencrypted messages). This is useful in scenarios where leaking information by broadcasting it in the clear is not an option (e.g., if the information allows to identify the party that leaked it). Thus we need to assume that the corrupted sender and receiver *do not share randomness or private communication channels*. An obvious implication is that they cannot corrupt the same party and they should only communicate through the bulletin board. In fact, this imposes much bigger limitations to their corruption abilities:

– They cannot corrupt parties that have parts of the key in common (e.g., in constructions relying on symmetric key cryptography), as in this case the common bits can be used as shared randomness.
– They cannot corrupt parties whose keys can be recovered from each other (as it is the case for public key cryptography, where usually the public key can be recovered from the secret key).
– Neither of them can have both read and write access to the board, otherwise they would have an (insecure but) two-way communication channel that would then allow for key-exchange. This means that the corrupted sender can only corrupt other senders, and analogously for the receiver. Moreover, corrupted senders should not have access to an encryption oracle, while corrupted receivers do: the first requirement is due to the fact that a corrupt sender could trivially break the property by "replaying" encryptions under keys of (honest) senders who are allowed to communicate to the corrupted receiver, while the latter is due to the fact that we want to model that receivers have access to the entire bulletin board, which may contain encryptions of known messages under keys of known identities.

The definition says that if a corrupted receiver can recover a message, then knowing some decryption keys did not help in the process. This is modeled by imposing that a party B without access to keys can recover the message with a similar success probability[3]. Remark that there is no consistency check on the ciphertext $\bar{s}$ output by the corrupted sender A_1: $\bar{s}$ could even be the entire view of A_1. In Sect. 7.2 we show that adding ciphertext verifiability yields stronger limitation on the communication between unauthorized parties.

Definition 3.3 (No Secret Write (NSW) Rule). *Let* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *be an adversary and consider the following game (oracles* $\mathcal{O}_G$ *and* $\mathcal{O}_E$ *are the key generation oracle and encryption oracle defined in Definition 3.2):*

[3] Alternatively one could require A_2 (and consequently B) to distinguish whether a ciphertext contains a subliminal message at all. This case is clearly implied by ours.

Experiments	
$\mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda,\mathcal{P})$	$\mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda,\mathcal{P})$
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda,\mathcal{P})$	$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda,\mathcal{P})$
$(\bar{m},\bar{s}) \leftarrow \mathsf{A}_1^{\mathcal{O}_G(\cdot,\mathsf{sen})}(pp)$	$(\bar{m},\bar{s}) \leftarrow \mathsf{A}_1^{\mathcal{O}_G(\cdot,\mathsf{sen})}(pp)$
$m' \leftarrow \mathsf{A}_2^{\mathcal{O}_G(\cdot,\mathsf{rec}),\ \mathcal{O}_E(\cdot)}(pp,\bar{s})$	$m'' \leftarrow \mathsf{B}^{\mathcal{O}_E(\cdot)}(pp,\bar{s})$
Return 1 *if* $\bar{m} = m'$,	*Return* 1 *if* $\bar{m} = m''$,
0 *otherwise.*	0 *otherwise.*

Let $\mathcal{Q}_1$ *(resp.,* $\mathcal{Q}_2$*) be the set of all queries* $q = (i,\mathsf{sen})$ *(resp.,* $q = (j,\mathsf{rec})$*) that* A_1 *(resp.,*A_2*) issues to* $\mathcal{O}_G$*. The adversary wins the experiment if* $m' = \bar{m}$ *while the following holds:*

No Communication Rule (NCR). $\forall\ (i,\mathsf{sen}) \in \mathcal{Q}_1,\ (j,\mathsf{rec}) \in \mathcal{Q}_2,\ \mathcal{P}(i,j) = 0.$

Given λ *and a policy* $\mathcal{P}$*, an ACE without sanitizer satisfies the No Secret Write rule if for all PPT* $\mathsf{A} = (\mathsf{A}_1,\mathsf{A}_2)$ *there exists a PPT algorithm* B *and a negligible function* negl *such that*

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda,\mathcal{P})\right] \geq \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda,\mathcal{P})\ \wedge\ \mathsf{NCR}\right] - \mathsf{negl}(\lambda).$$

Further remarks on defining NSWR security can be found in the full version, alongside the intuition behind the impossibility to instantiate an ACE without sanitization from symmetric key primitives.

4 Linear ACE Without Sanitizer from PKE

The first construction is akin to the original linear ACE from standard assumptions by Damgård et al. [4]. In such scheme, senders are given the public keys of all the receivers they are allowed to communicate with, and "decoy/placeholder" public keys for the receivers they are not allowed to communicated with (to make sure that ciphertexts generated by different senders have the same length). The encryption algorithm then encrypts the message under all the keys. The i-th receiver just decrypts the i-th ciphertext using its secret key. Sender's anonymity requires that ciphertexts do not leak any information about the key used to generate them, i.e., key indistinguishability [3].

Let $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a public key encryption scheme, that is IND-CPA secure[4] and IK-CPA. An ACE without sanitizer from PKE (denoted $\mathsf{ACE}^{\mathsf{pke}}$) can be instantiated as follow.

[4] It is enough that the PKE is IND-CPA, as whenever the receiver has to distinguish between the encryption of 2 different messages, it is not allowed to get the decryption key (as it would be in the Payload privacy game). In the sender anonymity game, when the adversary can ask for decryption keys, the only requirement is that is should be impossible to identify a sender from the encryption key it uses, which is guaranteed by the key-indistinguishability property.

Communication Model: parties communicate through a bulletin board. Only senders are allowed to write on the board. Receivers can only read from it.

Message Space: $\mathcal{M} := \{0,1\}^{\ell}$.

Setup: $(pp, msk) \leftarrow \mathsf{Setup}(1^{\lambda}, \mathcal{P})$

It generates the message set, and the number of parties n, of senders n_S, and of receivers n_R; all are included in pp, along with the policy. The master secret key is a list of $2n_R$ (distinct) pairs of asymmetric keys, i.e.,

$$msk = \left\{ ((pk_j^0, sk_j^0), (pk_j^1, sk_j^1)) \ : \ \begin{array}{c} \text{For } i = 1,2 \\ (pk_j^i, sk_j^i) \leftarrow \mathsf{PKE.KeyGen}(1^{\lambda}) \end{array} \right\}_{j \in [n_R]} .$$

Key Generation: $k_i \leftarrow \mathsf{KGen}(pp, msk, i, t)$

On input (i,t), the algorithm parses $msk = \{((pk_j^0, sk_j^0), (pk_j^1, sk_j^1))\}_j$, and behaves as it follows.

- If $i \neq 0$ and $t = \mathsf{sen}$, it returns a vector $ek_i = (ek_i[j])_{j \in [n_R]}$ such that $ek_i[j] \leftarrow pk_j^{\mathcal{P}(i,j)}$.
- If $i \neq 0$ and $t = \mathsf{rec}$, it returns $dk_i = sk_i^1$.
- If $i = 0$, returns $ek_0 = dk_0 = pp$.

Encryption: $c \leftarrow \mathsf{Enc}(pp, ek_i, m)$

Run $c_j \leftarrow \mathsf{PKE.Enc}(ek_i[j], m; \rho_j)$ for all $j \in [n_R]$ (ρ_j is a random string). Return $c = (c_j)_{j=1,\ldots,n}$.

Decryption: $m' \leftarrow \mathsf{Dec}(pp, dk_j, c)$

Let $c = (c_1, \ldots, c_{n_R})$. Return the output of $\mathsf{PKE.Dec}(dk_j, c_j)$ (which could be either a message m or $\perp$).

Theorem 4.1. *The* $\mathsf{ACE}^{\mathsf{pke}}$ *scheme is correct, and satisfies the properties of No-Read and No Secret Write as described in Sect. 3 if the public key encryption scheme is IND-CPA secure and key-indistinguishable.*

Proof. The proof is as follows.

Correctness. Correctness directly follows from the correctness of the PKE scheme.

No Read Rule. The No-Read Rule relies on both the IK-CPA and IND-CPA properties of the PKE scheme. The proof is deferred to the full version, as it closely follows the proof of [4, Theorem 3].

No Secret Write Rule. Given an adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ that wins the game $\mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda, \mathcal{P})$ with probability ϵ_{A}, to prove that the scheme satisfies the NSW Rule we need to construct an algorithm B that wins game $\mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda, \mathcal{P})$ with essentially the same probability (up to a negligible difference). Upon receiving $\bar{s}$ from A_1, the algorithm B runs A_2 internally on input $(pp, \bar{s})$ simulating the oracles as follows. First, it generates $2n_R$ pairs of PKE keys, and collects them in a fresh master secret key $\bar{msk} = \{((\bar{pk}_j^0, \bar{sk}_j^0), (\bar{pk}_j^1, \bar{sk}_j^1))\}_{j \in [n_R]}$. The oracles are then simulated using msk':

$\mathcal{O}_G$: on input (j, t) from A_2, B generates new keys according to the policy $\mathcal{P}$ using $\bar{msk}$. If $t = \mathsf{sen}$ B sends $ek_j = (\bar{pk}_i^{\mathcal{P}(j,i)})_{i\in[n_R]}$; if $t = \mathsf{rec}$, it sends $dk_j = \bar{sk}_j^1$. It stores the keys in a list $\mathcal{K}$.

$\mathcal{O}_E$: simulates the encryption oracle as specified in the security experiment using the appropriate key from $\mathcal{K}$.

Finally, B outputs the message that A_2 returns. By the definition of conditional probability, the success probability of B is

$$\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda, \mathcal{P})\right] = \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda, \mathcal{P}) \ \wedge \ \mathsf{NCR}\right] \cdot \mathsf{Pr}(\mathbf{E}),$$

where we denote by $\mathbf{E}$ the event "A_2 does not distinguish the simulated oracles from real ones". The only way A_2 could distinguish, is if the answers of the simulated oracles were inconsistent with the challenge ciphertext $\bar{s}$. However, in the real game the encryption keys queried by A_1 are statistically independent of the decryption keys queried by A_2, and they do not share state, thus any information encoded in $\bar{s}$ is statistically independent of the keys A_2 queries. The only way A_2 could get information about the encryption keys owned by A_1 would be by querying the encryption oracle on (i, m) for an i that A_1 has corrupted (such identity could be hardcoded in A_2, so the attack can be performed even in absence of shared state). If A_2 can distinguish that the ciphertext is not generated using the same key that A_1 received, then A can be exploited to break the key indistinguishability property of the PKE. Let q_E be the number of queries by A_2 to the encryption oracle. One can prove this by a sequence of hybrid games:

Game 0**.** This is the No-Secret-Write experiment.

Hybrid k **for** $0 \leq k \leq 2n_R$**.** In all hybrid games the view of A_1 is generated according to the NSWR experiment, i.e., using the master secret key msk generated at the beginning of the experiment. However, when generating the view of A_2, the challenger in Hybrid k generates the j-th key pairs in $\bar{msk}$ as follows:

Case $j < \lfloor k/2 \rfloor$**:** it generates fresh PKE key pairs $(\bar{pk}_j^0, \bar{sk}_j^0, \bar{pk}_j^1, \bar{sk}_j^1)$.

Case $j = k$**:** it generates a fresh key pair $(\bar{pk}, \bar{sk})$, and sets the j-th PKE pairs to be $(\bar{pk}, \bar{sk}, pk_j^1, sk_j^1)$ if k is odd and $(\bar{pk}_j^0, \bar{sk}_j^0, \bar{pk}, \bar{sk})$ if k is even, where $(pk_j^0, sk_j^0, pk_j^1, sk_j^1)$ is the j-th key pair in msk and $(\bar{pk}_j^0, \bar{sk}_j^0,) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.

Case $j > \lfloor k/2 \rfloor$**:** it uses the same PKE key pairs $(pk_j^0, sk_j^0, pk_j^1, sk_j^1)$ as in msk.

Let E_k be the event that A distinguishes between Hybrid k and Hybrid $k-1$. Lemma 4.2 shows that $\mathsf{Pr}(\mathrm{E}_k) \leq \frac{2}{3} q_E \epsilon_{\mathsf{ik\text{-}cpa}}(\lambda)$.

Game 1**.** This is the No-Secret-Write experiment as simulated by B. By definition, Hybrid 0 is exactly equal to Game 0, and Hybrid $2n_R$ is the same as Game 1. Therefore:

$$\mathsf{Pr}(\mathbf{E}) \geq 1 - 3n_R q_E \epsilon_{\mathsf{ik\text{-}cpa}}(\lambda),$$

where $\epsilon_{\mathsf{ik\text{-}cpa}}(\lambda)$ is the probability of breaking the IK-CPA property of the PKE scheme and $q_E = poly(\lambda)$ as A is a polynomial-time algorithm. $\square$

Lemma 4.2. *$\Pr(E_k) \leq 3\epsilon_{\mathit{ik\text{-}cpa}}(\lambda)q_E$ for all $k \in [2n_R]$.*

Proof. We split the proof in 3 cases:

Case 1: for all queries (j, sen) by A_1 to $\mathcal{O}_G$, $\mathcal{P}(j,k) = 0$.
Case 2: there are $\bar{i}, \bar{j} \in [n_S]$ such that A_1 queried $(\bar{i}, \mathsf{sen})$ and $(\bar{j}, \mathsf{sen})$ to $\mathcal{O}_G$ and $\mathcal{P}(\bar{i},k) = 1$, $\mathcal{P}(\bar{j},k) = 0$.
Case 3: for all queries (j, sen) by A_1 to $\mathcal{O}_G$, $\mathcal{P}(j,k) = 1$.

If $k = 0$ this is exactly the NSW experiment. If $k = 2n_R$ this is the NSW experiment as simulated by B. Assume now $0 < k < 2n_R$.

Let us start from k odd. In Case 1 A_1 sees pk_k^0 but not pk_k^1, while A_2 can query dk_k and receives sk_k^1 both in Hybrid k and in Hybrid $k-1$. The only difference is that in Hybrid k $\mathcal{O}_E$ uses $\bar{pk}_k, pk_k^1$ instead of pk_k^0, pk_k^1 as in Hybrid $k-1$. If A can distinguish in this case, we construct a PPT algorithm C that can win the IK-CPA experiment running A as a subroutine. C receives pk_0 and pk_1 from the IK-CPA experiment and generates msk setting $(pk_k^0, sk_k^0) = (pk_0, \perp)$ and $(\bar{pk}, \bar{sk}) = (pk_1, \perp)$. The rest of the master secret keys msk and $\bar{msk}$ are generated as specified by Hybrid k. Then it answers to $\mathcal{O}_G$ using msk for the queries by A_1 and $\bar{msk}$ for the queries by A_2. To answer queries from A_2 to $\mathcal{O}_E$, C selects a random $q \xleftarrow{\$} [q_E]$ and behaves as follows:

– C answers to the first $q-1$ queries using pk_0, pk_k^1 as the k-th encryption keys.
– When A_2 sends the q-th query (i, m), C returns m to the IK-CPA experiment and receives a challenge ciphertext $\bar{c}$. Then it generates the encryption of m as follows:

$$\begin{aligned}
c_j &\leftarrow \mathsf{PKE.Enc}(pk_j^{\mathcal{P}(i,j)}, m) \quad \text{for } j = 1, \dots, k-1 \\
c_j &\leftarrow \bar{c} \quad \text{for } j = k \text{ if } \mathcal{P}(i,k) = 0 \\
c_j &\leftarrow \mathsf{PKE.Enc}(pk_j^1, m) \quad \text{for } j = k \text{ if } \mathcal{P}(i,k) = 1 \\
c_j &\leftarrow \mathsf{PKE.Enc}(\bar{pk}_j^{\mathcal{P}(i,j)}, m) \quad \text{for } j = k+1, \dots, n_R
\end{aligned}$$

– C answers to the remaining $q_E - q + 1$ queries using pk_1, pk_k^1 as the k-th encryption keys.

Thus it follows that for k odd

$$\begin{aligned}
\Pr(\mathrm{E}_k \mid \text{Case 1}) &\leq |\Pr(\mathsf{A} \text{ wins Hybrid } k-1 \mid \text{Case 1}) - \Pr(\mathsf{A} \text{ wins Hybrid } k \mid \text{Case 1})| \\
&\leq |\sum_{Q=1}^{q_E} \Pr(\mathsf{A} \text{ wins the game } \mid q = Q \ \wedge \ \bar{c} \leftarrow \mathsf{PKE.Enc}(pk_0, m)) + \\
&\qquad - \Pr(\mathsf{A} \text{ wins the game } \mid q = Q \ \wedge \ \bar{c} \leftarrow \mathsf{PKE.Enc}(pk_1, m))| \\
&\leq q_E \epsilon_{\mathsf{ik\text{-}cpa}}(\lambda).
\end{aligned}$$

In Case 2 A_1 sees both pk_k^0 and pk_k^1, while A_2 cannot query dk_k both in Hybrid k and in Hybrid $k-1$. The difference is that in Hybrid k $\mathcal{O}_E$ uses

$\bar{pk}_k, pk^1_k$ instead of pk^0_k, pk^1_k as in Hybrid $k-1$. The reduction shown for Case 1 can be replicated without changes in this case. In Case 3 A_1 sees pk^1_k but not pk^0_k, while A_2 cannot query dk_k both in Hybrid k and in Hybrid $k-1$. Again the only difference is that in Hybrid k $\mathcal{O}_E$ uses $\bar{pk}_k, pk^1_k$ instead of pk^0_k, pk^1_k as in Hybrid $k-1$. Thus in this case the view of A in Hybrid k is statistically indistinguishable from the view of A in Hybrid $k-1$.

Finally, assume that k is even. In Case 1 A_1 sees pk^0_k but not pk^1_k, while A_2 can query dk_k and receives $\bar{sk}^1_k$ in Hybrid k and sk^1_k in Hybrid $k-1$. The encryption oracle $\mathcal{O}_E$ uses $\bar{pk}_k^{\,0}, \bar{pk}$ in Hybrid k, and $\bar{pk}^0_k, pk^1_k$ in Hybrid $k-1$. As the adversary does not see pk^1_k, the view of A in Hybrid k is statistically indistinguishable by the view of A in Hybrid $k-1$. In Case 3 A_1 sees pk^1_k but not pk^0_k, while A_2 cannot query dk_k both in Hybrid k and in Hybrid $k-1$. The difference is that in Hybrid k $\mathcal{O}_E$ uses $\bar{pk}_k^{\,0}, \bar{pk}$ instead of $\bar{pk}^0_k, pk^1_k$ as in Hybrid $k-1$. The previous reduction can be adapted to this case as follows. C receives pk_0 and pk_1 from the IK-CPA experiment and generates msk setting $(pk^1_k, sk^1_k) = (pk_0, \perp)$ and $(\bar{pk}, \bar{sk}) = (pk_1, \perp)$. The rest of the master secret keys msk and $\bar{msk}$ are generated as specified by Hybrid k. Then it answers to $\mathcal{O}_G$ using msk for the queries by A_1 and $\bar{msk}$ for the queries by A_2. To answer queries from A_2 to $\mathcal{O}_E$, C selects a random $q \xleftarrow{\$} [q_E]$ and behaves as follows:

- C answers to the first $q-1$ queries using pk_0, pk^1_k as the k-th encryption keys.
- When A_2 sends the q-th query (i, m), C returns m to the IK-CPA experiment and receives a challenge ciphertext $\bar{c}$. Then it generates the encryption of m as follows:

$$\begin{aligned}
c_j &\leftarrow \mathsf{PKE.Enc}(pk_j^{\mathcal{P}(i,j)}, m) \quad \text{for } j = 1, \ldots, k-1 \\
c_j &\leftarrow \mathsf{PKE.Enc}(\bar{pk}^0_j, m) \quad \text{for } j = k \text{ if } \mathcal{P}(i,k) = 0 \\
c_j &\leftarrow \quad \bar{c} \quad \text{for } j = k \text{ if } \mathcal{P}(i,k) = 1 \\
c_j &\leftarrow \mathsf{PKE.Enc}(\bar{pk}_j^{\mathcal{P}(i,j)}, m) \quad \text{for } j = k+1, \ldots, n_R
\end{aligned}$$

- C answers to the remaining $q_E - q + 1$ queries using pk_1, pk^1_k as the k-th encryption keys.

Analogously to the case of k odd, for k even it holds that

$$\mathsf{Pr}(\mathrm{E}_k \mid \text{Case } 3) \leq q_E \epsilon_{\mathsf{ik\text{-}cpa}}(\lambda).$$

Finally, in Case 2 A_1 sees both pk^0_k and pk^1_k, while A_2 cannot query dk_k both in Hybrid k and in Hybrid $k-1$. The difference is again that in Hybrid k $\mathcal{O}_E$ uses $\bar{pk}_k^{\,0}, \bar{pk}$ instead of $\bar{pk}^0_k, pk^1_k$ as in Hybrid $k-1$. The reduction shown for Case 1 can be replicated without changes in this case. Therefore, for all k it holds that

$$\mathsf{Pr}(\mathrm{E}_k) = \sum_{i=1}^{3} \mathsf{Pr}(\text{Case } i)\mathsf{Pr}(\mathrm{E}_k \mid \text{Case } i) \leq 3 q_E \epsilon_{\mathsf{ik\text{-}cpa}}(\lambda).$$

□

5 Compact ACE from Hybrid Encryption

The previous construction has the problem that the length of ciphertexts depends linearly on $\ell \cdot n_R$. This can be improved using a hybrid encryption technique: combining $\mathsf{ACE}^{\mathsf{pke}}$ with a rate-1 symmetric key encryption (SKE) scheme yields a more compact ACE (denoted by $\mathsf{ACE}^{\mathsf{he}}$), which outputs ciphertexts whose size scales with $\ell + n_R$ instead. Interestingly, there is no known analogous hybrid encryption version of the original construction of [4].

Let $(\mathsf{SE.KeyGen}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ be a rate-1 symmetric encryption scheme that is $\mathsf{lor\text{-}cpa}$ secure, and let $\mathsf{ACEnoS} = (\mathsf{ACE.Setup}, \mathsf{ACE.KGen}, \mathsf{ACE.Enc}, \mathsf{ACE.Dec})$ be an ACE without sanitizer that is NRR and NSWR secure.

Communication Model: parties communicate through a bulletin board; senders and receivers have write-only and read-only access respectively.
Message Space: $\mathcal{M} := \{0,1\}^\ell$.
Setup: $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$
Return $(pp, msk) \leftarrow \mathsf{ACE.Setup}(1^\lambda, \mathcal{P})$.
Key Generation: $k_i \leftarrow \mathsf{KGen}(pp, msk, i, t)$
Return $k_i \leftarrow \mathsf{ACE.KGen}(pp, msk, i, t)$.
Encryption: $c \leftarrow \mathsf{Enc}(pp, ek_i, m)$
Generate a one-time secret key $sk \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$, and encrypt the message using it: $c_1 \leftarrow \mathsf{SE.Enc}_{sk}(m)$. Then encrypt the key using the ACEnoS: $c_2 \leftarrow \mathsf{ACE.Enc}(pp, ek_i, sk)$. Return $c = (c_1, c_2)$.
Decryption: $m' \leftarrow \mathsf{Dec}(pp, dk_j, c)$
Parse $c = (c_1, c_2)$. Decrypt the secret key $sk' \leftarrow \mathsf{ACE.Dec}(pp, dk_j, c_2)$. If $sk' = \perp$, return $\perp$. Else return $m' \leftarrow \mathsf{SE.Dec}_{sk'}(c_1)$.

Efficiency, Storage Requirements, and Optimizations. The length of the ciphertext is $\mathrm{O}(n_R + \ell)$ using a rate-1 SKE. The full version contains a construction from predicate encryption that outputs more compact ciphertexts (of length $\mathrm{O}(\log(n_S) + \lambda)$) when instantiated for policies such that $\min_{j \in [n_R]} S_j = \mathrm{O}(\log n_S)$ where S_j is the set of senders allowed to communicate with the receiver j.

Theorem 5.1. *The protocol previously described is correct, and satisfies the properties of No-Read and No Secret Write if the SKE is* $\mathsf{lor\text{-}cpa}$ *secure, and* ACEnoS *satisfies correctness, NRR and NSWR as described in Sect. 3.*

The security proof is akin to that of $\mathsf{ACE}^{\mathsf{pke}}$, and is deferred to the full version.

6 Game-Specific Obfuscation

We suggest a variant of obfuscation that is weaker that Virtual Blackbox (VBB) obfuscation and hence may be possible to implement in general. VBB obfuscation requires that the obfuscated program gives nothing to the receiver, other than oracle access to the original program, and it is well known that no obfuscator can be capable of VBB-obfuscating every program.

Experiment $\mathsf{Exp}^{G^0}_{\mathsf{A},\mathsf{Obf}}(\lambda, \mathcal{F}, q)$	**Experiment** $\mathsf{Exp}^{G^1}_{\mathsf{B},\mathsf{Obf}}(\lambda, \mathcal{F}, q)$
$st \leftarrow (\lambda, \mathcal{F}, q)$	$st \leftarrow (\lambda, \mathcal{F}, q)$
$(pp, k) \leftarrow \mathsf{C}(0; st)$	$(pp, k) \leftarrow \mathsf{C}(0; st)$
For $i = 1, \ldots, q$:	For $i = 1, \ldots, q$:
$\quad z_i^{\mathsf{A}} \leftarrow \mathsf{A}^{\mathsf{C}(1,k,\cdot;st),\mathsf{C}(3,\cdot;st)}(pp)$	$\quad z_i^{\mathsf{B}} \leftarrow \mathsf{B}^{\mathsf{C}(2,k,\cdot,\cdot;st),\mathsf{C}(3,\cdot;st)}(pp)$
$b \leftarrow \mathsf{C}(4, z_1^{\mathsf{A}}, \ldots, z_q^{\mathsf{A}}; st)$	$b \leftarrow \mathsf{C}(4, z_1^{\mathsf{B}}, \ldots, z_q^{\mathsf{B}}; st)$
Return b.	Return b.

Fig. 1. Security experiment for Game-Specific obfuscation.

Here, we consider instead a security game G (formalized as an experiment in Fig. 1), in which a challenger C plays against an adversary A, using an obfuscator Obf. The game comes with a specification of a family of programs $\mathcal{F} := \{\mathsf{P}_{k,\mathsf{p}}\}_{k \in \{0,1\}^\lambda, \mathsf{p} \in \{0,1\}^m}$, parameterized by k and by a label p of some length m, so we have one member of the family for each pair (k, p). This is meant to cover a wide range of applications where obfuscated programs may be used: very often, an application bakes one or more cryptographic keys into the program, this is modelled by the parameter k. The label p is useful in a multiparty scenario, where parties may be given programs that depend on their identity, for instance.

The game proceeds in rounds, where in each round of the game, A can query C on various labels p to obtain obfuscated programs $\hat{\mathsf{P}}_k^{\mathsf{p}} = \mathsf{Obf}(\mathsf{P}_k^{\mathsf{p}})$, as well as for other data (such as public parameters). At the end of each round, A returns some final output z_i, which is remembered between rounds. Optionally, the game may allow A to remember additional state information between rounds (not represented in Fig. 1). In the end, C decides if A won the game. Our definition compares this to a similar experiment where, however, the adversary B only gets oracle access to the programs.

Importantly, C can decide not to answer a query, based on the label and its current state. This models conditions one would typically have in a game, such as: the game models a scheme with several parties participating, some of which are corrupt, and the adversary is not allowed to query a program that was given to an honest player. Since the same C plays against both A and B, they are under the same constraints, so B cannot "cheat" and make queries that A could not.

For simplicity, we let C choose a single parameter k initially. We can easily generalize to cases where programs using several different values of k are used.

Definition 6.1 (Game-Specific Obfuscation). *We say that* Obf *is a* game-specific secure obfuscator *(GSO) relative to G and $\mathcal{F}$ if for every PPT adversary* A*, there exists a PPT adversary* B *which plays G using only oracle access to each obfuscated program, and where* $|\mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{G^0}_{\mathsf{A},\mathsf{Obf}}(\lambda, \mathcal{F}, q)] - \mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{G^1}_{\mathsf{B},\mathsf{Obf}}(\lambda, \mathcal{F}, q)]|$ *is negligible, where the challenger behaves as follows:*

Challenge Generation: *on input* $(0; (\lambda, \mathcal{F}, q))$*, it returns* $k \in \{0,1\}^\lambda$ *and some general public parameters pp.*

Program Obfuscation: *on input* $(1, k, \mathsf{p}; st)$*, it returns the obfuscation* $\hat{\mathsf{P}}^{\mathsf{p}}_k \leftarrow \mathsf{Obf}(\mathsf{P}^{\mathsf{p}}_k)$ *of the program, or* $\perp$*.*

Oracle Access to Programs: *on input* $(2, (k, \mathsf{p}, m); st)$ *it returns the evaluation of the program* $\mathsf{P}^{\mathsf{p}}_k(m)$*, or* $\perp$*.*

Other Data: *on input* $(3, \cdot; st)$ *it can return additional data.*

Winning Condition Check: *on input* $(4, z_1, \ldots, z_q; st)$ *it returns* 1 *if the adversary won the game, 0 otherwise.*

Every mode of operation can update the state st of the challenger too, if required by the game.

Note that this definition, while implied by VBB, makes a much weaker demand than VBB: we assume that the obfuscation gives nothing more than oracle access, only as far as winning G is concerned, and the obfuscator only needs to obfuscate programs in $\mathcal{F}$. Indeed, the impossibility result for VBB does not apply to game-specific obfuscation in general, it just rules out its existence for a specific game and family of programs. The notion is somewhat incomparable to iO obfuscation: obfuscators secure in the iO sense are usually claimed to be able to obfuscate any program, and can potentially be applied in any security game, but on the other hand, iO only guarantees indistinguishability between programs with the same input/output behavior. Even when restricting to assume the existence of iO/GSO for specific programs (as it happens in constructions relying on iO), still iO and GSO target different aspects: GSO has no specific requirement on the family of programs, while iO needs them to compute the same function; on the other hand, iO still guarantees indistinguishability for every game, while GSO targets a specific one.

As usual, we also require the obfuscators to *preserve functionality* (the input-output behaviour of the obfuscated program is equivalent to the original program) and *polynomial slowdown* (the obfuscated program should can at most be polynomially slower/larger than the original one).

7 ACE with Ciphertext Verifiability

In this section we explore whether it is possible to obtain more than just preventing parties from establishing secure subliminal channels. The intuition is that it should be possible to restrict corrupted parties in the bandwidth of their subliminal channels by adding some form of *ciphertext verifiability* to our model. Ciphertext verifiability allows any party with access to the bulletin board to verify that ciphertexts appended to the public board have been generated honestly and according to policy, *even if the party is not allowed to decrypt them by the policy.* We then show a scheme that allows to restrict the bandwidth of corrupted senders to logarithmic in the security parameter under a novel variant of obfuscation, namely the GSO introduced in the previous section. We find this a promising indication that public verification can help to restrict subliminal communication between corrupted parties. As a byproduct, we get a construction whose complexity only scales polylogarithmically with the number of parties.

7.1 Ciphertext Verifiability

Parties, policies and the communication model are the same as in Sect. 3. The difference is that an ACE with ciphertext verifiability (VACE) is composed by 5 algorithms ($\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Verify}, \mathsf{Dec}$). The verification algorithm Verify allows receivers to verify that ciphertexts published in the bulletin board are well-formed according to their decryption key:

Verification $b \in \{0,1\} \leftarrow \mathsf{Verify}(dk_j, c)$
On input a ciphertext c and a decryption key, the algorithm outputs 1 if $c \leftarrow \mathsf{Enc}(pp, ek_i, m)$ for some (unknown) honestly generated sender's key ek_i and message $m \in \mathcal{M}$, and 0 otherwise.

Remark that the definition implies that verification can be done using dk_0, i.e., the decryption key of the receiver with identity $j = 0$ which by policy cannot receive messages[5]. Differently from ACEnoS, now dk_0 might not be equal to the public parameters (while ek_0 still is). Moreover, dk_0 is not part of them: it is given only to receivers, not to the senders. This follows quite naturally from the communication model: as senders have write-only access to the public board, they cannot see (thus verify) ciphertexts by other senders than themselves[6].

The introduction of such algorithm requires to modify the properties of security and correctness as well. This new construction of ACE should satisfy both correctness as defined in Sect. 3 and a completeness requirement (i.e., that honestly generated ciphertexts pass verification).

Definition 7.1 (Completeness). *A VACE scheme is complete if for all $\lambda, m \in \mathcal{M}$, $i \in [n_S]$, $j \in [n_R]$ it holds*

$$\Pr\left[1 \leftarrow \mathsf{Verify}(dk_j, \mathsf{Enc}(pp, ek_i, m)) \ : \ \begin{array}{r} (pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P}), \\ ek_i \leftarrow \mathsf{KGen}(pp, msk, i, \mathsf{sen}), \\ dk_j \leftarrow \mathsf{KGen}(pp, msk, j, \mathsf{rec}) \end{array} \right] = 1,$$

where the probabilities are taken over the random coins of all the algorithms.

To ensure that verification is meaningful, the outcome of verification should be consistent when done with different keys.

[5] The inclusion of the identity 0 for senders and receivers with no rights is standard in normal access control encryption, cf. [4].

[6] In fact, it seems to be necessary for a more technical reason related to the NSWR (as the verification key could be seen as shared randomness between corrupted senders and receivers).

Definition 7.2 (Verification Consistency). *Given a policy $\mathcal{P}$, a* VACE *scheme verifies consistently if, for every PPT adversary* A *there exists a negligible function* negl *such that*

$$\Pr\left[b_0 \neq b_1 \;\middle|\; \begin{array}{l} (pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P}) \\ (i_0, i_1, c) \leftarrow \mathsf{A}^{\mathcal{O}_G(\cdot,\cdot)}(pp) \\ \textit{For } k = 0, 1 \\ \quad dk_{i_k} \leftarrow \mathsf{KGen}(pp, msk, i_k, \mathsf{rec}) \\ \quad b_k \leftarrow \mathsf{Verify}(dk_{i_k}, c) \end{array} \right] \leq \mathsf{negl}(\lambda),$$

where the $\mathcal{O}_G$ returns ek_j on input (j, sen), and dk_j on input (j, rec).

The No Read Rule remains unchanged as such property is not concerned with enforcing the policy, but with the anonymity and privacy of the scheme. On the other hand, the winning condition of the No Secret Write Rule changes to impose that the challenge ciphertext successfully verifies w.r.t. some fixed receiver key. This, combined with consistency of verification (which we just defined) implies that a successful verification w.r.t. even just the public verification key dk_0 is enough to guarantee it w.r.t. all receiver keys (which could be impossible to check efficiently in the game if the number of receivers is superpolynomial).

The verification key dk_0 is only given to the corrupted receiver A_2 and to the public verifier B but not to the corrupted sender A_1, as the latter cannot read from the public board, but just write on it.

Definition 7.3 (No Secret Write Rule). *Let* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *be an adversary and consider the following game:*

Experiments	
$\mathsf{Exp}^{\mathsf{nusw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda, \mathcal{P})$	$\mathsf{Exp}^{\mathsf{nusw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda, \mathcal{P})$
$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$ $(\bar{m}, c) \leftarrow \mathsf{A}_1^{\mathcal{O}_G(\cdot,\mathsf{sen})}(pp)$ $m' \leftarrow \mathsf{A}_2^{\mathcal{O}_G(\cdot,\mathsf{rec}),\ \mathcal{O}_E(\cdot)}(pp, c)$ *Return* 1 *if* $\quad \bar{m} = m' \ \wedge\ 1 \leftarrow \mathsf{Verify}(dk_0, c), dk_0 \leftarrow \mathcal{O}_G(0, \mathsf{rec})$, 0 *otherwise.*	$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$ $(\bar{m}, c) \leftarrow \mathsf{A}_1^{\mathcal{O}_G(\cdot,\mathsf{sen})}(pp, \bar{m})$ $m'' \leftarrow \mathsf{B}^{\mathcal{O}_E(\cdot),\mathcal{O}_G(0,\mathsf{rec})}(pp, c)$ *Return* 1 *if* $\quad \bar{m} = m'' \ \wedge\ 1 \leftarrow \mathsf{Verify}(dk_0, c), dk_0 \leftarrow \mathcal{O}_G(0, \mathsf{rec})$, 0 *otherwise.*

Oracles	
$\mathcal{O}_G(j, t)$: *If* $\exists\ k_j$ s.t. $(k_j, j, t) \in \mathcal{L}$, *return* k_j *Else* $k_j \leftarrow \mathsf{KGen}(pp, msk, j, t)$ $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{(k_j, j, t)\}$ *Return* k_j.	$\mathcal{O}_E(j, m)$: $ek_j \leftarrow \mathcal{O}_G(j, \mathsf{sen})$ *Return* $c \leftarrow \mathsf{Enc}(pp, ek_j, m)$.

Let $\mathcal{Q}_1$ (resp., $\mathcal{Q}_2$) be the set of all queries $q = (i, \mathsf{sen})$ (resp., $q = (j, \mathsf{rec})$) that A_1 (resp.,A_2) issues to $\mathcal{O}_G$. The adversary wins the experiment if $m' = \bar{m}$ and the ciphertext verifies while the following holds:

No Communication Rule *(*NCR*).* $\forall\ (i, \mathsf{sen}) \in \mathcal{Q}_1,\ (j, \mathsf{rec}) \in \mathcal{Q}_2$, *it should hold that $\mathcal{P}(i, j) = 0$.*

Given λ and a policy $\mathcal{P}$, a ACE without sanitizer with verifiable ciphertexts satisfies the No Secret Write rule if for all PPT $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *there exists a PPT algorithm* B *and a negligible function* negl *such that*

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda, \mathcal{P})\right] \geq \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda, \mathcal{P}) \ \wedge \ \mathsf{NCR}\right] - \mathsf{negl}(\lambda).$$

Ciphertext Verifiability vs. Sanitization. It is fair to wonder whether adding public verifiability yields an ACE with sanitization. This is not the case because: (1) the sanitizer/verification key is public; (2) in the VACE case, behavior of sanitizer/verifier is checkable by other receivers; (3) the access structure to a public board usually requires an authentication layer: verification (and possible identification of dishonest senders) can be enforced in that layer.

7.2 VACE from Game Specific Obfuscation

Verifiability of a ciphertext means that any party can verify that the ciphertext satisfies some relation, i.e., that *has some structure*, which bounds the entropy of the ciphertext. While this is not enough to prevent subliminal channels completely (as this seems to require the injection of true randomness, e.g., cf. [4,8]), in this section we show that this is enough to meaningfully restrict the bandwidth of corrupted senders.

We build a VACE following the IND-CCA PKE construction from iO by Sahai and Waters [9], with the following changes: (1) we impose that every ciphertext encrypts the identity of the sender in addition to the message, and (2) decryption is done by an obfuscated program that checks the policy too. As the original protocol outputs ciphertexts composed by two parts, the encryption of the message and a value used as authentication/integrity check, we easily get a VACE construction that is NRR secure assuming iO with a proof similar to [9]. However, proving NSWR from iO seems impossible, thus we rely on a GSO assumption on the obfuscator (further details in Sect. 7.3).

We now consider messages to be just one bit, i.e., $\mathcal{M} = \{0, 1\}$, and assume that $n_S = \mathsf{poly}(\lambda)$ (as this is needed when using the puncturable PRF in the proof of the NRR rule).

Setup: $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$
Generate the keys for the PRFs: $K_k \xleftarrow{\$} \{0,1\}^\lambda$, $k = 1, 2$. The algorithm returns $pp = (\lambda, \mathcal{P}, \mathcal{M})$ and the master secret key $msk = (K_1, K_2)$.

Key Generation: $k_i \leftarrow \mathsf{KGen}(pp, msk, i, t)$
Generate the obfuscated circuits $\hat{\mathsf{P}}^s_i \leftarrow \mathsf{Obf}(\lambda, \mathsf{P}^s_i)$, $i \in [n_S] \setminus \{0\}$, and $\hat{\mathsf{P}}^r_i \leftarrow \mathsf{Obf}(\lambda, \mathsf{P}^r_i)$, $i \in [n_R]$, of the programs P^s_i and P^r_j in Fig. 2, padded so that they are as long as the programs in the reductions (cf. proof of Theorem 7.4 and 7.7).

- If $i \neq 0$ and $t = \mathsf{sen}$, return $ek_i = (\hat{\mathsf{P}}^s_i)$.
- If $i = 0$ and $t = \mathsf{sen}$, return $ek_0 = pp$.
- If $t = \mathsf{rec}$, return $dk_i = (\hat{\mathsf{P}}^r_i)$.

```
$\mathsf{P}^s_i(m, s)$

$t \leftarrow \mathsf{PRG}(s)$
$cipher \leftarrow \mathsf{PRF}_1(K_1, t) \oplus (m, i)$
$sig \leftarrow \mathsf{PRF}_2(K_2, (t \mid\mid cipher))$
$c \leftarrow (t, cipher, sig))$
Return $c$.
```

```
$\mathsf{P}^r_j(c)$

Parse $c = (t, cipher, sig)$.
$b \leftarrow (sig == \mathsf{PRF}_2(K_2, (t \mid\mid cipher)))$
If $(b == 1)$
    $(m, i) \leftarrow cipher \oplus \mathsf{PRF}_1(K_1, t)$
    If $(\mathcal{P}(i, j) == 1)$
        Return $(1, m)$
    Else return $(1, \perp)$.
Else Return $(0, \perp)$.
```

Fig. 2. Encryption and decryption programs.

Encryption: $c \leftarrow \mathsf{Enc}(pp, ek_i, m)$
Sample $s \xleftarrow{\$} \{0,1\}^\lambda$. Return $c \leftarrow \hat{\mathsf{P}}^s_i(m, s)$.
Decryption: $m' \leftarrow \mathsf{Dec}(pp, dk_j, c)$
Run $(b, m') \leftarrow \hat{\mathsf{P}}^r_j(c)$ and return m'.
Verification: $b \in \{0,1\} \leftarrow \mathsf{Verify}(dk_j, c)$
Run $(b, m') \leftarrow \hat{\mathsf{P}}^r_j(c)$ and return b.

For ease of exposition we split the security proof of the VACE in two theorems, as the NRR and NSWR require different assumptions on Obf. In particular Theorem 7.4 shows NRR security and only requires the standard notion of iO, whereas Theorem 7.7 uses the novel GSO assumption. Note that one could also have chosen to prove the NRR security of the VACE assuming GSO instead of iO, but we opted for using the minimal assumptions for each theorem.

Theorem 7.4. *The* VACE *previously defined satisfies correctness and completeness, and has consistent verification, assuming the correctness of its building blocks. In addition, if* Obf *is a iO and* PRF_1, PRF_2 *and* PRG *are two puncturable PRF and a PRG respectively, the previous* VACE *and is NRR secure.*

The proof of Theorem 7.4 relies on the standard techniques introduced in [9], and is deferred to the full version. Remark that verification consistency follows easily from the fact that the first bit of the output of $\hat{\mathsf{P}}^r_j$ is independent of the value of j, thus it is the same for all $j \in [n_R]$.

7.3 No Secret Write Rule of VACE

We argue that indistinguishability obfuscation does not seem enough to prove NSWR security for our VACE. A major hint in this direction is that proving that the NSWR holds seems to require to show that A_2 cannot distinguish the real experiment from an experiment where both the encryption oracle and the receiver keys are simulated using only the information available to B (i.e., the encryption oracle and the verification key). However, we do not see a way to simulate the decryption keys that preserves their I/O behavior without knowing the master secret keys. Such a consistency in the I/O behavior of the keys is needed because A_1 could still transmit information to A_2 related to the behavior of the senders' keys queried by A_1, e.g., the output on a particular input (s, m): as B does not know which keys have been queried by A_1, it cannot rely on the encryption oracle to answer these queries consistently. However, simulation can be done assuming Obf to be a secure GSO. In particular, the obfuscator is assumed to be GSO secure for the following family of programs and game.

Definition 7.5 **($\mathcal{F}$).** *The family $\mathcal{F} = \{\mathsf{P}_{k,\mathsf{p}}\}_{k,\mathsf{p}}$ contains all the possible keys:*

- *$k = msk = (K_1, K_2)$, and*
- *$\mathsf{p} = (j, t)$ is the identity and type of party, and*
- *$\mathsf{P}_{k,(j,t)} = \mathsf{P}^t_j$, $t \in \{s, r\}$ as defined in Fig. 2.*

Definition 7.6 **(G_{nsw}).** *The game G_{nsw} runs for $q = 2$ rounds and is played by a challenger $\mathsf{C}_{\mathsf{nsw}}$ that behaves as follows:*

- *$\mathsf{C}(0, \ldots; st)$ returns $(pp, k) = (pp, msk) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{P})$ and stores them in st (alongside a round counter).*
- *$\mathsf{C}(1, (k, (j, t)); st)$ returns the output of $\mathsf{KGen}(pp, msk, j, t)$ and stores the query in a list q_i for $i = 1, \ldots, q$ in st.*
- *$\mathsf{C}(2, (k, (j, t), m); st)$ it returns the evaluation of the program $\mathsf{P}^t_j(m)$.*
- *$\mathsf{C}(3, st)$ returns $\perp$ during round 1, and $\bar{s}$ in round 2.*
- *$\mathsf{C}(4, z_1, z_2; st)$ parses $z_1 = (\bar{m}, \bar{s})$ and returns 1 if the following three conditions hold:*
  1. *$z_2 = \bar{m}$*
  2. *$1 \leftarrow \mathsf{Verify}(dk_0, \bar{s})$*
  3. *q_1 (resp., q_2) contains only queries for sender (resp., receiver) keys, and for every $(i, \mathsf{sen}) \in q_1$ and $(j, \mathsf{rec}) \in q_2$ it holds that $\mathcal{P}(i, j) = 0$.*

Note that we have chosen to only use the GSO assumption where it is necessary, namely the NSWR property. Therefore, since the NRR property is still proven using the iO assumption, the PRFs used in the construction are still puncturable even if this property is not explicitly used in the proof of the NSWR property.

Theorem 7.7. *Assuming Obf is a secure GSO for $\mathcal{F}$ and G_{nsw} as in Definition 7.5 and 7.6, and given two puncturable PRF and a PRG, the previous VACE is NSWR secure. Moreover, assuming that only ciphertexts that pass the verification are posted, it only allows for subliminal channels of bandwidth at most $O(\log(\lambda))$.*

Proof. Proving the NSWR relies on the hypothesis that Obf is a secure GSO for $(\mathcal{F}, G_{\mathsf{nsw}})$. Indeed, the NSWR experiment in Definition 7.3 is exactly equal to the game in Fig. 1 where $(\mathcal{F}, G_{\mathsf{nsw}})$ are as in Definition 7.5 and 7.6, and the adversary A in the GSO experiment behaves like A_1 in the first round, and like A_2 in the second. This implies that the probability that $(\mathsf{A}_1, \mathsf{A}_2)$ win the NSWR experiment is the same as the probability that A wins the GSO experiment. In fact, from GSO security it follows that for any adversary A there exists a second adversary A' that has only oracle access to all the keys, but wins the game G_{nsw} (i.e., the NSWR experiment) with almost the same probability:

$$\begin{aligned}\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{A}_2)}(\lambda,\mathcal{P}) \ \wedge \ \mathsf{NCR}\right] &= \Pr\left[1 \leftarrow \mathsf{Exp}^{G^0_{\mathsf{nsw}}}_{\mathsf{A},\mathsf{Obf}}(\lambda,\mathcal{F},2)\right] \\ &\leq \Pr\left[1 \leftarrow \mathsf{Exp}^{G^1_{\mathsf{nsw}}}_{\mathsf{A}',\mathsf{Obf}}(\lambda,\mathcal{F},2)\right] + \epsilon_{GSO}. \qquad (1)\end{aligned}$$

Let us now analyze the winning probability of A'. Denote by $(\mathsf{A}'_1, \mathsf{A}'_2)$ the execution of A' in the first and second round of G^1_{nsw} respectively. This adversary now does not receive the sender (or receiver) keys, but is only given oracle access to them. In fact, the oracle only evaluates the plaintext version of the keys, thus it is possible to substitute the PRF and PRG used in the keys with random functions, without significantly impacting the winning probability of A':

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{G^1_{\mathsf{nsw}}}_{\mathsf{A}',\mathsf{Obf}}(\lambda,\mathcal{F},2)\right] = \Pr\left[1 \leftarrow \mathsf{Exp}^{G^2_{\mathsf{nsw}}}_{\mathsf{A}',\mathsf{Obf}}(\lambda,\mathcal{F},2)\right] + \epsilon_\rho, \qquad (2)$$

where ϵ_ρ is the probability of distinguishing the PRF and PRG from a random function, and the game G^2_{nsw} is a modification of G^1_{nsw} where $\mathsf{C}(2,\cdot)$ answers the queries executing the code of P^t_i where PRF_1, PRF_2 and PRG have been substituted by random functions.

At this point we can already observe that the bandwidth of the subliminal channel (for ciphertexts that pass the verification) has to be at most $\mathrm{O}(\log(\lambda))$. Indeed, in game G^2_{nsw} the components t and *cipher* of the ciphertext are uniformly random while the tag *sig* is deterministically generated from them, thus a corrupted sender is restricted to encode a subliminal message in a ciphertext only through rejection sampling: A'_1 can only try encrypting randomly chosen messages (the ciphertext does not reveal anything about the plaintext, thus it is fair to assume that the subliminal message and the plaintext are independently chosen) until the ciphertext finally encodes the subliminal message. If the sender runtime is restricted to be polynomial-time, this limits the amount of rejection sampling that it can do, restricting the amount of information encoded in the ciphertext (the subliminal message) to $\mathrm{O}(\log(\lambda))$. The GSO assumption allows to conclude the argument: as A_1 cannot do (much) better than A'_1, the adversary can send short subliminal messages in the real experiment too.

Finally, we conclude the proof of the NSWR by showing an algorithm B that can win the NSWR experiment running A'_2 internally, with almost the same probability as the adversary A. Recall that in game G^2_{nsw} the sender keys oracle (i.e., the simulated $\mathsf{C}(2,(\cdot,(\cdot,\mathsf{sen}),m);st)$, which can be called in both rounds) returns a uniformly random bit string. This in particular implies that the view of A'_1 (i.e., A' at round 1) and A'_2 are independent of the master secret key *msk* generated at the beginning of the game. Therefore, a simulator B can win the NSWR experiment running A' at round 2 internally by generating a fresh pp', msk' for the VACE and use them to simulate $\mathsf{C}(2,\cdot)$. As the public parameters of the scheme only contain λ, $\mathcal{M}$ and $\mathcal{P}$, there is no way for A'_2 to distinguish between the real and simulated experiment, and it holds that

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{G^2_{\mathsf{nsw}}}_{\mathsf{A}',\mathsf{Obf}}(\lambda,\mathcal{F},2)\right] = \Pr\left[1 \leftarrow \mathsf{Exp}^{G^2_{\mathsf{nsw}}}_{(\mathsf{A}'_1,\mathsf{B}),\mathsf{Obf}}(\lambda,\mathcal{F},2)\right] = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{nsw}}_{(\mathsf{A}_1,\mathsf{B})}(\lambda,\mathcal{P})\right]. \qquad (3)$$

Combining Eqs. (1), (2), and (3) yields the claim. □

On the Need of Ciphertext Verifiability. Ciphertext verifiability is crucial for the previous argument to go through: if one cannot verify that the ciphertext has been generated by the obfuscated program, a corrupted sender could just set the ciphertext to be the (subliminal) message it wants to send. So long as the data structure of the ciphertext fits the specifications, the subliminal channel cannot be detected. The next lemma (which is a folklore result) shows that our result is optimal: stricter restrictions on the subliminal channel require sanitization.

Lemma 7.8. *Let λ be the security parameter. An encoding scheme* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *(either symmetric or asymmetric, deterministic or probabilistic) such that the domain of* Enc_k *has dimension at least* $\mathsf{poly}(\lambda)$ *for every k output by the key generation* KG *always allows for insecure subliminal channels with bandwidth* $O(\log(\lambda))$ *(in absence of a trusted sanitization step) assuming the adversary runs in polynomial-time and has* oracle access *to the encryption algorithm.*

Proof. Consider an encoding $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ that satisfies basic correctness (reportend in the following for completeness):

Correctness: $\forall \lambda \in \mathbb{N}$, $\forall m \in \mathcal{M}$, $\exists \epsilon = \mathsf{negl}(\lambda) \ : \ \mathsf{Pr}(m' \neq m \mid (k_e, k_d) \leftarrow \mathsf{KG}(1^\lambda),\ c \leftarrow \mathsf{Enc}(k_e, m),\ m' \leftarrow \mathsf{Dec}(k_d, c)) \leq \epsilon$.

Then a PPT adversary A_1 that has only *oracle access* to the encryption algorithm can transmit any subliminal message $\bar{m}$ such that $|\bar{m}| \leq \mathrm{O}(\log(\lambda))$ to a PPT receiver A_2 by sending a single valid ciphertext, even in the worst case scenario in which A_2 cannot decrypt the ciphertext, nor it shares state with A_1, and independently of the security guarantees of the encoding scheme.

The attack is very simple. Having oracle access to the encoding algorithm means that on input m, the oracle returns its encryption under a key fixed at the beginning of the game (and unknown to A_1). The adversary A_1 can query the encryption oracle to obtain $q = \mathsf{poly}(\lambda)$ *distinct* ciphertexts $\{c_i\}_{i=1,\ldots,q}$ (because it runs in polynomial-time, and the domain of the encryption algorithm is large enough for the ciphertexts to be distinct). As they are all distinct, there exists w.h.p. a ciphertext c_i whose (w.l.o.g.) first $|\bar{m}|$ bits are equal to $\bar{m}$. □

Acknowledgments. Cecilia Boschini is supported by the Università della Svizzera Italiana under the SNSF project number 182452, and by the Postdoc.Mobility grant No. P500PT_203075. Claudio Orlandi is supported by: the Concordium Blockhain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC).

References

1. Alwen, J., Shelat, A., Visconti, I.: Collusion-free protocols in the mediated model. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 497–514. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_28
2. Barak, B., et al.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_1
3. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_33
4. Damgård, I., Haagh, H., Orlandi, C.: Access control encryption: enforcing information flow with cryptography. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 547–576. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_21
5. Fuchsbauer, G., Gay, R., Kowalczyk, L., Orlandi, C.: Access control encryption for equality, comparison, and more. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10175, pp. 88–118. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_4
6. Kim, S., Wu, D.J.: Access control encryption for general policies from standard assumptions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 471–501. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_17
7. Lu, Y., Ciampi, M., Zikas, V.: Collusion-preserving computation without a mediator. To appear at CSF 2022
8. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 657–686. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_22
9. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC (2014)
10. Simmons, G.J.: The prisoners' problem and the subliminal channel. In: Chaum, D. (eds.) Advances in Cryptology. Springer, Boston, MA (1984). https://doi.org/10.1007/978-1-4684-4730-9_5