Nonce-Based Cryptography: Retaining Security When Randomness Fails

Mihir Bellare and Björn Tackmann(✉)

Department of Computer Science and Engineering,
University of California, San Diego, USA
{mihir,btackmann}@eng.ucsd.edu
http://cseweb.ucsd.edu/~mihir/
http://cseweb.ucsd.edu/~btackmann/

Abstract. We take nonce-based cryptography beyond symmetric encryption, developing it as a broad and practical way to mitigate damage caused by failures in randomness, whether inadvertent (bugs) or malicious (subversion). We focus on definitions and constructions for nonce-based public-key encryption and briefly treat nonce-based signatures. We introduce and construct hedged extractors as a general tool in this domain. Our nonce-based PKE scheme guarantees that if the adversary wants to violate IND-CCA security then it must do *both* of the following: (1) fully compromise the RNG (2) penetrate the sender system to exfiltrate a seed used by the sender.

1 Introduction

An old security adage says there is no point putting strong locks on the door if you leave the window open. The lock here is modern public-key encryption, proven to meet the strong IND-CCA goal. The window is the assumption made in these proofs that, at every encryption, the encryptor has access to perfect, fresh (independent from prior) randomness. To allow encryption to fulfill in practice the promise it makes in theory, we must close the window. This paper develops nonce-based public-key encryption as a practical way to do this. It goes on to develop nonce-based digital signatures.

Randomness. That randomness failures occur and lead to cryptographic failures is by now very well known and does not need to be belabored. The news of interest is perhaps that it is getting worse. Let us explain. There are two sources of randomness failures. The first, which has been with us a while and is not going away, is *bugs*. A good example is the Debian Linux vulnerability present from September 2006 to May 2008 where a programmer removed some lines of code from the OpenSSL source, resulting in there being only 15 bits of entropy in the seed for the PRNG [1]. HDWH [19] finds cryptographic vulnerabilities in a significant fraction of TLS and SSH servers due to what they call "malfunctioning RNGs." And the list goes on. The second source of randomness failures, which may have been with us for a while but of which we have learned only recently due

M. Fischlin and J.-S. Coron (Eds.): EUROCRYPT 2016, Part I, LNCS 9665, pp. 729–757, 2016.
DOI: 10.1007/978-3-662-49890-3_28

to the mass-surveillance revelations, is *subversion*, the deliberate and targeted attempt to weaken randomness. At the South by South West (SXSW) 2014 conference, Snowden said "we know that the encryption algorithms we are using today work... it is the random number generators that are attacked as opposed to the encryption algorithms themselves." The prime example here is Dual EC, a RNG the NSA designed to have a backdoor and then pushed into standards and adoption. The ability to compromise security in practice via the backdoor has been demonstrated in [14].

Prior work. The basic definitions of security for public-key encryption (PKE), namely IND-CPA [18] and IND-CCA [5,15], provide no guarantees if the randomness is bad. There is a long line of work giving new definitions of security that do provide such guarantees, and building PKE schemes to meet these definitions.

The simplest way to avoid vulnerabilities due to poor randomness is to not use randomness at all. Deterministic PKE [3,6,12,17] however only provides security when the messages have high min-entropy. This limits utility (for example, we may want to encrypt votes, which have low min-entropy) and, although in a different context, CGPR [13] show that in practice the entropy of "real" data is often quite low. Hedged PKE [4,7,27] extends Deterministic PKE to provide privacy as long as the message and randomness together have sufficient min-entropy. This is a significant benefit and we recommend that one use Hedged PKE whenever possible. But the limitations remain. Since messages regularly do not in fact have entropy [13], and the "randomness" can be entirely predictable (this happened both with the Debian Linux bug and the Dual EC subversion), the message and randomness together still may not have enough entropy for hedged encryption to provide security. A further limitation of both Deterministic and Hedged PKE is that security is only provided for messages that do not depend on the public key. (This second limitation can be partially addressed but at some cost [26].) Yilek [32] defines and achieves security against randomness-reset attacks, where the randomness is perfect but the adversary can force its re-use across different encryptions. This is useful in the context of virtual machine resets but not more broadly. PSS [24] introduce related-randomness attacks, where encryption is under adversary-specified functions of some initial uniform randomness. However, negative results they provide show that for many functions one cannot achieve security.

In summary, all these notions have some limitations and the practical benefit they provide is not clear. Most importantly, these were all designed in the older mindset of RNG failures due to bugs and can break down severely when the RNG is subverted. The latter is the new reality against which we need to defend.

Nonces. Rogaway [28,29] introduced nonce-based symmetric-key encryption, where the encryption algorithm is deterministic, taking the shared key, message and a quantity called a nonce. Security is provided as long as the nonce does not repeat. The notion was strengthened by Rogaway and Shrimpton (RS) [30]. Rogaway suggests that packet sequence numbers may play the role of the nonce. Motivations he provides include reducing implementor error and

achieving stronger notions of security. We suggest that nonces can be used much more broadly and are a good defense against poor randomness. The goal we pursue in depth is nonce-based PKE.

The obvious extension fails. Towards this we begin by noting that a direct extension of nonce-based symmetric-key encryption as defined in [28–30] to the public-key setting does not work. Such an extension would have the encryption algorithm $\mathcal{E}$ be deterministic, taking input the encryption key ek, message m and a nonce n to return the ciphertext $C = \mathcal{E}(ek, m, n)$. The privacy game would give the adversary an oracle that takes messages m_0, m_1 and a nonce n (in the definitions of [28–30], nonces are adversarially chosen subject to not repeating) to return $C = \mathcal{E}(ek, m_b, n)$, where b is a challenge bit chosen at random by the game. Security would require that the adversary has little advantage in guessing b. But such security would not be achieved because in the public-key setting the adversary has ek and can itself encrypt. Thus it could query its oracle with any m_0, m_1, n of its choice to get back ciphertext $C = \mathcal{E}(ek, m_b, n)$, and itself compute $C_0 = \mathcal{E}(ek, m_0, n)$. If $C = C_0$ it knows that the challenge b is 0, else it is 1.

Nonce-based PKE. We define a nonce-based PKE scheme NPE as follows. The receiver runs key-generation algorithm $\mathsf{NPE.Kg}$ as usual to get an encryption key ek and decryption key dk. Not as usual, the sender begins by locally running a *seed generation* algorithm $\mathsf{NPE.sKg}$ to get a *seed* xk. The encryption algorithm $\mathsf{NPE.Enc}$ is *deterministic*, taking in addition to the usual ek and message m, two new inputs, a nonce n and the *seed* xk, and returning ciphertext $C = \mathsf{NPE.Enc}(ek, xk, m, n)$. Decryption is unchanged, taking dk and a ciphertext C to return a message. The receiver does *not* need to know the seed, and the keys and seed are entirely independent. The sender can either re-use the same seed across multiple encryptions, or generate a fresh one at every encryption, or anything in between, and the receiver will be oblivious to all of this.

Security is captured via two games and corresponding requirements. Nonce-based privacy One (NBP1) asks that IND-CCA privacy be maintained as long as message-nonce pairs do not repeat. That is, the only way security fails is if, for the same message, a nonce is re-used. This is a very strong guarantee. However there is one caveat, namely this holds when the seed is kept private from the adversary. Nonce-based privacy Two (NBP2) addresses the possibility that the adversary compromises the sender's system and obtains the seed. Even in this case, it guarantees IND-CCA privacy as long as nonces are unpredictable to the adversary. The formalizations are in terms of a stateful *nonce generator* NG that takes an adversary specified input η to return the next nonce, so that nonces are (indirectly) under adversary control.

In practice we would expect a combination of a variety of things to be used as the nonce, for example the current time (this does not repeat) and prior ciphertexts, but, also, randomness from the system RNG, since, for NBP2, nonce unpredictability is required. (This is a departure from the symmetric setting.) However guarantees in the face of poor randomness are much better than before, as we now explain.

What this buys us. Intuitively our definitions are saying the following. Consider two cases. The expected and good case is that the sender seed stays private. In this case we get IND-CCA privacy *regardless of the quality of the randomness*, the only requirement being that message-nonce pairs do not repeat. The latter is a mild condition, such repetition being unlikely with reasonable nonces, even simply using the date and time as the nonce. The other case is that the sender's system is compromised and the seed is exposed. In this case, we are effectively in the setting of standard PKE and we cannot deterministically provide IND-CCA. We guarantee that we do no worse than standard PKE, meaning we provide IND-CCA as long as the randomness (here part of the nonce) is good. But in fact we do better, since the requirement on nonces is only unpredictability. This means that we are safe even if the outputs of the RNG are correlated and structured, as long as they remain unpredictable.

Put another way, if a subvertor wanted to compromise privacy, it would not suffice to compromise the RNG. They would have to also break in to the sender's system, find the seed, and exfiltrate it. Frequent rotation of seeds (which has effectively no cost) makes this even harder. This ups the ante. Now, it is true that with the NSA's capabilities in malware, we should not under-estimate their ability to penetrate a target sender. But this would have to be done on per-sender basis, making mass surveillance harder.

Hedged extractors and our scheme. It is easy to achieve either of NBP1 or NBP2 in isolation. We can get an NBP1 nonce-based PKE scheme by encrypting under a conventional (randomized) IND-CCA PKE scheme with the coins set to the result of a PRF keyed by the sender seed and applied to the message and nonce, but there is no reason this scheme would also be NBP2 secure. We can get an NBP2 nonce-based PKE scheme by encrypting under the conventional IND-CCA PKE scheme with the coins set to the result of an extractor keyed by the seed and applied to the (message and) nonce, but there is no reason this scheme would also be NBP1 secure. To simultaneously get both properties, we introduce and use *hedged extractors*.

A hedged extractor HE takes a seed (also called a key) xk, a message m and a nonce n to deterministically return a string $r = \mathsf{HE}(xk, (m, n))$. It has two properties: (1) It is a PRF, meaning if xk is random and hidden then the outputs look random even to an adversary that picks m, n, and (2) it is an extractor, meaning if xk is random but known, then r looks random if (m, n) is unpredictable (meaning, has enough min-entropy). Again, achieving either goal in isolation is trivial. The task is to achieve them simultaneously, in the same construction. We give two solutions, one in the ROM, the other in the standard model. The first is easy but practical, and likely to be what we would use, namely to simply apply the RO to $xk, (m, n)$. The second combines a PRF with a strong randomness extractor via XOR. The ROM solution delivers optimal security, the standard-model one a bit less due to the inherent limitations of strong randomness extractors, namely that they are only guaranteed to work for seed-independent inputs that retain min-entropy even conditioned on prior inputs.

Our nonce-based PKE scheme is then simply defined via the same paradigm as for the in-isolation cases, namely we encrypt under a conventional (randomized) IND-CCA PKE scheme with the coins set to the result of a hedged extractor keyed by the sender seed and applied to the message and nonce. Both NBP1 and NBP2 security are proven for this scheme assuming IND-CCA of the conventional scheme and security of the hedged extractor.

Discussion and pragmatics. We can view nonce-based cryptography as moving the traditional abstraction boundary between cryptography and system RNGs closer to the cryptography. The view is that, in the presence of bad RNGs, a safer and better architecture is that the cryptography take on as large a share of the burden of providing security as possible, in other words, rely on its environment as little as possible. Our suggestion here is that the environment is relied on only to produce nonces with relatively weak requirements. This view is in some ways the opposite of that represented by work that aims to strengthen RNGs against failure or subversion [16]. In practice the two can co-exist and their combined presence will increase security.

Our nonce-based encryption scheme is simple and modular, a way to transform any given conventional IND-CCA scheme into an NBP1+NBP2 secure nonce-based scheme. With a practical choice of hedged extractor such as our ROM one, we retain the efficiency attributes of the initial PKE scheme. In our scheme, decryption is unchanged. The decryptor does not need to change its software or even know that nonces are being used. These attributes make it easier to deploy nonce-based PKE as a practical defense against poor randomness.

In the above-discussed prior work aimed at increasing resistance of PKE to randomness failures, the *model* was unchanged in the sense that the object whose security was being considered continued in syntax to be a classical public-key encryption scheme as per [18]. Nonce-based encryption is a new model (because the sender has a seed) and a new syntax (there is a seed generation algorithm and the encryption algorithm is different). It is these changes, and in particular not just the nonce, but the combination of nonce and seed, that are a game changer and result in significantly better guarantees against poor randomness compared to prior work.

Picking a seed, like picking a key, does require (good) randomness. The viewpoint here, as in all the prior work discussed above, is that there is a difference between static and dynamic randomness usage. We assume good randomness for key generation because effort can be invested in it. Current key-generation software often has the user generate coins by waving their mouse around. Good seed generation would require similar effort, but one would expect to use a seed for some time so this effort is not frequent. This is flexible. If a seed is lost due to a system reboot, or compromised, the user can elect to make the effort to pick a new one. If you are encrypting from multiple systems (your desktop, laptop and phone) each can have its own, independently chosen seed.

Nonce-based signatures and beyond. We define nonce-based signatures, where the signing algorithm is deterministic, and takes not only the signing key and message, but also a seed and nonce. We require that (1) if the seed remains

hidden then we have regular security (unforgeability) regardless of how nonces are generated, and (2) if the seed is exposed, then we have security as long as the nonces were unpredictable. Section 5 formalizes this and shows how to convert any signature scheme into a nonce-based one with these security properties using a hedged extractor.

Due to their speed and short signature sizes, the most attractive signature schemes for practice are elliptic-curve versions of DSA, El Gamal and Schnorr [31]. However, they are randomized, and fail spectacularly when the randomness is bad. Discussions on the `cfrg` forum show overwhelming support for making these schemes deterministic. This is easily done, by deriving the coins either as a PRF, keyed by a seed that is part of the secret key and applied to the message, or as a RO applied to the secret key and message [9,21,23,25], and the popular Ed25519 signature scheme of [11] already embodies this. Making a scheme nonce-based complements this traditional de-randomization, retaining the benefits of deterministic signing while adding further ones. See Sect. 5 for more extensive background and discussion.

Nonces in combination with seeds can similarly be used in many other areas of cryptography to provide resilience in the face of poor randomness or even provide other gains. Our work aims to be illustrative rather than exhaustive.

Related work. BKS [8] introduce stateful PKE. Here also the sender can maintain a seed. They show that this leads to significant efficiency gains. Their schemes are however randomized, and there are no nonces. An interesting direction for future work is to combine their methods with ours to get similar efficiency gains for nonce-based PKE.

Rogaway [29] discusses nonces as "surfacing the IV." As motivation, he says that when IVs are implicit, implementors and even books get things wrong. He says that often nonces are readily available, for example packet sequence numbers. He does not seem to explicitly mention robustness in the face of randomness failure as a goal in the symmetric case. Intriguingly, in the final section of the paper, he goes on to say: " ... it makes just as much sense to consider nonce-based public-key encryption schemes as it does to consider nonce-based symmetric encryption schemes. This provides an approach to effectively weakening the requirement for randomness on the sender." Our work has pursued this suggestion. It is surprising that this waited 12 years.

2 Notation and Standard Definitions

Notation. We let ε denote the empty string. If X is a finite set, we let $x \xleftarrow{\$} X$ denote picking an element of X uniformly at random and assigning it to x. Algorithms may be randomized unless otherwise indicated. Running time is worst case. If A is an algorithm, we let $y \leftarrow A(x_1, \ldots; r)$ denote running A with random coins r on inputs $x_1, \ldots$ and assigning the output to y. We let $y \xleftarrow{\$} A(x_1, \ldots)$ be the result of picking r at random and letting $y \leftarrow A(x_1, \ldots; r)$. We let $[A(x_1, \ldots)]$ denote the set of all possible outputs of A when invoked with inputs $x_1, \ldots$. We use the code based game playing framework of [10].

Game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})$	Game $\mathbf{G}^{\mathrm{ind}}_{\mathsf{PE}}(\mathcal{A})$	Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}}(\mathcal{A})$
$fk \leftarrow\!\!\$\ \mathsf{F.Keys}$	$(ek, dk) \leftarrow\!\!\$\ \mathsf{PE.Kg}$	$(sk, vk) \leftarrow\!\!\$\ \mathsf{DS.Kg}$
$c \leftarrow\!\!\$\ \{0,1\}$	$b \leftarrow\!\!\$\ \{0,1\}\ ;\ C \leftarrow \emptyset$	$M \leftarrow \emptyset$
$c' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{FN,RO}}$	$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{ENC,DEC}}(ek)$	$(m, s) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{SIG}}(vk)$
Return $(c = c')$	Return $(b = b')$	$v \leftarrow (\mathsf{DS.Ver}(vk, m, s) = 1)$
FN(x)	ENC(m_0, m_1)	Return $(v \wedge (m \notin M))$
If $(c = 1)$ then	If $(\lvert m_0\rvert \neq \lvert m_1\rvert)$ then return $\perp$	SIG(m)
$\quad S[x] \leftarrow \mathsf{F}^{\mathrm{RO}}(fk, x)$	$c \leftarrow\!\!\$\ \mathsf{PE.Enc}(ek, m_b)$	$s \leftarrow \mathsf{DS.Sig}(sk, m)$
Else	$C \leftarrow C \cup \{c\}$	$M \leftarrow M \cup \{m\}$
$\quad$ If $S[x] = \perp$ then	Return c	Return s
$\quad\quad S[x] \leftarrow\!\!\$\ \mathsf{F.Rng}$	DEC(c)	
Return $S[x]$	If $(c \in C)$ then return $\perp$	
RO(x, l)	$m \leftarrow \mathsf{PE.Dec}(ek, dk, c)$	
If $T[x, l] = \perp$ then	Return m	
$\quad T[x, l] \leftarrow\!\!\$\ \{0,1\}^l$		
Return $T[x, l]$		

Fig. 1. Games for defining PRF security of a function family F, standard IND-CCA security of a standard PKE scheme PE and EUF-CMA security of a signature scheme DS.

(See Fig. 1 for an example.) By $\Pr[\mathrm{G}]$ we denote the event that the execution of game G results in the game returning true. Random oracles are variable output length, represented by a game procedure RO that takes x, l and returns a random string of length l. The min-entropy of a random variable X over $\mathcal{X}$ is defined as $\mathrm{H}_\infty(\mathsf{X}) = -\log(\max_{x \in \mathcal{X}}(\Pr[\mathsf{X} = x]))$.

Function families. A family of functions $\mathsf{F}\colon \mathsf{F.Keys} \times \mathsf{F.Dom} \to \mathsf{F.Rng}$ is a two-argument function that takes a key K in the key space $\mathsf{F.Keys}$, an input x in the domain $\mathsf{F.Dom}$ and returns an output $\mathsf{F}(K, x)$ in the range $\mathsf{F.Rng}$. In the ROM, F takes an oracle RO.

Pseudo-random functions. The security of F as a PRF is defined via game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})$ that is associated to adversary $\mathcal{A}$ and shown in Fig. 1. Here F could have access to a RO and thus the game is in the ROM. Tables S, T are assumed initially $\perp$ everywhere. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathcal{A}) = 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})] - 1$.

Public-key encryption. A *public-key encryption scheme* PE specifies the following. Receiver key-generation algorithm $\mathsf{PE.Kg}$ returns an encryption key ek and associated decryption key dk. Encryption algorithm $\mathsf{PE.Enc}$ takes ek and message $m \in \{0,1\}^*$ to return a ciphertext c. Deterministic decryption algorithm $\mathsf{PE.Dec}$ takes ek, dk and ciphertext c to return a value in $\{0,1\}^* \cup \{\perp\}$, and we require standard decryption correctness. The advantage of an adversary $\mathcal{A}$ in breaking the IND-CCA security of PE is defined as $\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{PE}}(\mathcal{A}) =$

$2\Pr[\mathbf{G}^{\mathrm{ind}}_{\mathsf{PE}}(\mathcal{A})] - 1$ for the game $\mathbf{G}^{\mathrm{ind}}_{\mathsf{PE}}(\mathcal{A})$ described in Fig. 1. This represents a conventional (not nonce-based scheme), and thus PE.Enc is randomized. We will use such schemes as base schemes and we will need to surface their coins, writing $c \leftarrow \mathsf{PE.Enc}(ek, m; r)$ to mean that PE.Enc is run with coins r to deterministically return c. The length of the coins is denoted PE.rl.

Digital signatures. A digital signature scheme DS specifies the following. Signer key-generation algorithm DS.Kg returns a signature key sk and a verification key vk. Signing algorithm DS.Sig takes sk and message $m \in \{0,1\}^*$ to return a signature $s \in \{0,1\}^{\mathsf{DS.ol}}$. Verification algorithm DS.Ver takes vk, message $m \in \{0,1\}^*$, and signature $s \in \{0,1\}^{\mathsf{DS.ol}}$, to return a bit $b \in \{0,1\}$. The advantage of an adversary $\mathcal{A}$ in breaking the EUF-CMA security of DS is defined as $\mathbf{Adv}^{\mathsf{uf}}_{\mathsf{DS}}(\mathcal{A}) = \Pr[\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}}(\mathcal{A})]$ for the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}}(\mathcal{A})$ described in Fig. 1. Again we may need to surface the coins $r \in \{0,1\}^{\mathsf{DS.rl}}$ of DS.Sig, writing $s \leftarrow \mathsf{DS.Sig}(sk, m; r)$.

3 Hedged Extractors

Our nonce-based schemes work simply by supplying coins to a base scheme via a *hedged extractor* keyed by the sender seed. This primitive, that we introduce and build here, is a function family that has two security properties. The first is that it is a PRF. The second, which we define and call ror (real or random) security, formalizes randomness of outputs when the key (seed) is known. Clearly this can only be achieved with some restrictions, and the type of ror security achieved will vary across constructions, from the "best possible" achieved by our ROM construction to a weaker, but we think still meaningful, version for our standard model construction. To make the goals precise we first introduce the notion of a nonce generator.

Nonce generators. A nonce generator is an algorithm NG that, on input a *nonce selector* η and a current state St, returns a nonce n, belonging to the range set $\mathsf{NG.Rng} \subseteq \{0,1\}^*$ of NG, together with an updated state, written $(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$. We say the generator has nonce length $\mathsf{NG.nl} \in \mathbb{N}$ if $\mathsf{NG.Rng} = \{0,1\}^{\mathsf{NG.nl}}$. Let $\mathcal{P}$ be an adversary called a predictor and consider game $\mathbf{G}^{\mathrm{pred}}_{\mathsf{NG}}(\mathcal{P})$ of Fig. 2. Let

$$\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(\mathcal{P}) = \Pr[\mathbf{G}^{\mathrm{pred}}_{\mathsf{NG}}(\mathcal{P})] \quad \text{and} \quad \mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(q_1, q_2) = \max_{\mathcal{P}} \mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(\mathcal{P}) \ ,$$

where the maximum is over all $\mathcal{P}$ making at most $q_1 \in \mathbb{N}$ queries to GEN and $q_2 \in \{0,1\}$ queries to EXPOSE. Now let us explain. A call to GEN generates the next nonce in the sequence, returning nothing to the adversary, The adversary can influence the choice of nonces through its choice of the selector η. The EXPOSE oracle allows to additionally get access to the state of the nonce generator. To win, the adversary needs to guess some generated nonce or create a collision between generated nonces.

Game $\mathbf{G}^{\mathrm{pred}}_{\mathsf{NG}}(\mathcal{P})$	Game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$
$St \leftarrow \varepsilon$; $s \leftarrow 0$; $N \leftarrow \emptyset$	$St \leftarrow \varepsilon$; $xk \leftarrow\!\!\$\ \mathsf{HE.Keys}$; $c \leftarrow\!\!\$\ \{0,1\}$; $s \leftarrow 0$
$n \leftarrow\!\!\$\ \mathcal{P}^{\mathrm{GEN},\mathrm{EXPOSE}}$	$c' \leftarrow\!\!\$\ \mathcal{G}^{\mathrm{RoR},\mathrm{EXPOSE},\mathrm{RO}}(xk)$
Return $((n \in N)$ OR $\mathsf{coll})$	Return $(c = c')$
GEN(η)	RoR(m, η)
If $s = 1$ then return $\perp$	If $s = 1$ then return $\perp$
$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$	$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$
If $(n \in N)$ then $\mathsf{coll} \leftarrow \mathsf{true}$	If $(c = 1)$ then $r \leftarrow \mathsf{HE}^{\mathrm{RO}}(xk, (m, n))$
$N \leftarrow N \cup \{n\}$	Else $r \leftarrow\!\!\$\ \mathsf{HE.Rng}$
EXPOSE	Return r
$s \leftarrow 1$	EXPOSE
Return St	$s \leftarrow 1$; Return St
	RO(x, l)
	If $T[x, l] = \perp$ then $T[x, l] \leftarrow\!\!\$\ \{0,1\}^l$
	Return $T[x, l]$

Fig. 2. Games for defining predictability of the nonce generator NG and real-or-random security of function family HE.

Nonce generators represent another departure from nonce-based symmetric encryption. In the latter the adversary picks the nonce, but we saw in Sect. 1 that this does not work in the public-key setting. Instead, we model the process of a sender picking a nonce via a nonce generator.

In discussions, we refer to NG as weakly unpredictable if it is unpredictable for adversaries making no EXPOSE query, meaning $\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(q_1, 0)$ is "small" for "practical" values of q_1, and strongly unpredictable if it is unpredictable even for adversaries making an EXPOSE query, meaning $\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(q_1, 1)$ is "small" for "practical" values of q_1. If NG is strongly unpredictable it is also weakly unpredictable, but not necessarily vice versa. That is, the class of weakly unpredictable nonce generators is larger than the class of strongly unpredictable nonce generators.

Real or random security. Let $\mathsf{HE}\colon \mathsf{HE.Keys} \times \mathsf{HE.Dom} \to \mathsf{HE.Rng}$ be an oracle family of functions (this means it may have access to a random oracle). The first input is referred to as the "key" or the "seed." The domain has the form $\mathsf{HE.Dom} = \{0,1\}^* \times \mathsf{HE.NS}$, so that an input is a pair of strings, the first referred to as the "message" and the second as the "nonce," the latter drawn from a nonce space $\mathsf{HE.NS}$ associated to HE. Consider game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$ of Fig. 2 associated to HE, nonce generator NG and an adversary $\mathcal{G}$. The number of queries to EXPOSE is either 0 or 1, and the number to other oracles is arbitrary. Let

$$\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) = 2\Pr[\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})] - 1\ .$$

Note that here the adversary is given the key (seed) xk as input, unlike in the PRF notion, modeling exposure of the sender seed. Security asks that outputs of $\mathsf{HE}(xk, (\cdot, \cdot))$, for adversary-chosen messages and nonces from the nonce generator, are indistinguishable from random. Clearly, this will be possible only with certain restrictions, which will emerge when we discuss our constructions below.

In game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$, we say that adversary $\mathcal{G}$ is *agnostic* if its RoR queries do not depend on the seed xk. More formally, there exists a pair $(\mathcal{G}_1, \mathcal{G}_2)$ of algorithms such that $\mathcal{G}^{\mathrm{RoR,EXPOSE,RO}}(xk)$ does the following:

$$St \leftarrow\!\!\$\ \mathcal{G}_1^{\mathrm{RoR,EXPOSE,RO}}(\varepsilon)\ ;\ c' \leftarrow \mathcal{G}_2^{\mathrm{EXPOSE,RO}}(xk, St)\ ;\ \text{Return } c'\ .$$

This represents one of the restrictions we will impose to achieve ror security in the standard model.

Hedged extractors. A *hedged extractor* HE: $\mathsf{HE.Keys} \times \mathsf{HE.Dom} \to \mathsf{HE.Rng}$ is an oracle family of functions as above where the goal is that (1) HE is a PRF, meaning $\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{A})$ is low for any adversary $\mathcal{A}$ of practical resources, and also (2) $\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$ is small for some class of nonce generators NG and some class of ror adversaries $\mathcal{G}$, both specified via results for individual hedged extractors. We give a ROM construction and standard model one. Both achieve PRF security, but differ in the type of ror security achieved. The ROM construction achieves ror security for unpredictable nonce generators (both weak and strong) and for all ror adversaries. This is "best possible" because the unpredictability assumption is easily seen to be necessary. The standard model construction achieves ror security for strongly unpredictable generators and agnostic ror adversaries. These restrictions reflect limitations of the randomness extractors that are our underlying tool. The restriction to agnostic adversaries reflects that randomness extractors only work on seed-independent distributions, and the strong unpredictability requirement on the generator reflects that when extracting from a sequence of inputs, one needs not only that each has some min-entropy, but that it does even given the others.

ROM hedged extractor. We start by giving a simple and efficient construction **HE1** of a hedged extractor in the ROM. Let ℓ be a desired number of output bits for the extractor, and k a desired seed (key) length. Associated to ℓ, k is the hedged extractor $\mathsf{HE} = \mathbf{HE1}[\ell, k]$: $\{0,1\}^k \times (\{0,1\}^* \times \{0,1\}^*) \to \{0,1\}^\ell$ defined by

$$\mathsf{HE}^{\mathrm{RO}}(xk, x) = \mathrm{RO}((xk, x), \ell)\ .$$

Here $\mathsf{HE.Keys} = \{0,1\}^k$, $\mathsf{HE.Dom} = \{0,1\}^* \times \mathsf{HE.NS}$ with $\mathsf{HE.NS} = \{0,1\}^*$, and $\mathsf{HE.Rng} = \{0,1\}^\ell$.

The following lemma states that this construction achieves PRF security and also achieves real or random security assuming only that the nonce generator is unpredictable. Note the latter only requires each nonce to individually be unpredictable, but nonces may be arbitrarily correlated, and it could be that given n_1 one can easily predict n_2. But the extractor works nonetheless.

Lemma 1. *Let $\ell, k \geq 1$ be integers and let $\mathsf{HE} = \mathbf{HE1}[\ell, k]$ be the ROM function family associated to ℓ and k as above.*

1. *If $\mathcal{A}$ is an adversary making q_2 queries to its* RO *oracle, then*

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{A}) \le \frac{q_2}{2^k} \ . \tag{1}$$

2. *Let* NG *be a nonce generator. If $\mathcal{G}$ is an adversary making q_1 queries to its* RoR *oracle, q_2 queries to its* RO *oracle, and $q_3 \in \{0,1\}$ queries to its* EXPOSE *oracle, then*

$$\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) \le q_2 \cdot \mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(q_1, q_3) \ . \tag{2}$$

Note that in the 2nd case, the reduction preserves the number of EXPOSE queries, meaning the number reflected by q_3 is the number made by $\mathcal{G}$. This is the best one could hope for.

Proof (Lemma 1). For the proof of Eq. 1, consider the games $\mathrm{G}_0, \mathrm{G}_1$ of Fig. 3, where G_1 contains the boxed code and G_0 does not. Letting c denote the challenge bit in game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{A})$, the following, justified below, establishes Eq. 1:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{A}) &= \Pr[\,\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{A}) \,|\, c = 1\,] - \Big(1 - \Pr[\,\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{A}) \,|\, c = 0\,]\Big) && (3)\\
&= \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0] && (4)\\
&\le \Pr[\mathrm{G}_0 \text{ sets } \mathsf{bad}] && (5)\\
&\le \frac{q_2}{2^k} \ . && (6)
\end{aligned}$$

Equation 3 is a standard re-formulation of the definition of the advantage. In game G_0, replies to queries to the Fn and RO oracles are independently distributed, so that it is equivalent to the $c = 0$ case of game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{A})$, up to the flipping of the outcomes from true to false. In game G_1, the replies to Fn queries are given by $\mathsf{HE}^{\mathrm{RO}}$, making it equivalent to the $c = 1$ case of game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{A})$. This justifies Eq. 4. Games $\mathrm{G}_0, \mathrm{G}_1$ are identical until bad (differ only in statements following the setting of bad to true), so the Fundamental Lemma of Game Playing [10] justifies Eq. 5. In game G_0, replies to all oracle queries are random and independent of xk so the probability that the latter is queried as part of an RO query is at most the quantity of Eq. 6.

For Eq. 2, consider the games $\mathrm{G}_2, \mathrm{G}_3$ of Fig. 3, where G_3 contains the boxed code and G_2 does not. Let predictor adversary $\mathcal{P}$ be as specified in Fig. 3. Letting c denote the challenge bit in game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$, the following, justified below, establishes Eq. 2:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) &= \Pr[\,\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) \,|\, c = 1\,] - \big(1 - \Pr[\,\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) \,|\, c = 0\,]\big) && (7)\\
&= \Pr[\mathrm{G}_3] - \Pr[\mathrm{G}_2] && (8)\\
&\le \Pr[\mathrm{G}_2 \text{ sets } \mathsf{bad}] && (9)\\
&\le q_2 \cdot \mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(\mathcal{G}) && (10)\\
&\le q_2 \cdot \mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(q_1, q_3) \ . && (11)
\end{aligned}$$

Game G_0 / $\boxed{\mathrm{G}_1}$

$xk \leftarrow\$ \{0,1\}^k$; $c' \leftarrow\$ \mathcal{A}^{\mathrm{FN},\mathrm{RO}}$; Return $(c' = 1)$

$\mathrm{FN}(w)$

If $S[w] = \bot$ then $S[w] \leftarrow\$ \{0,1\}^\ell$; Return $S[w]$

$\mathrm{RO}(w, l)$

If $T[w, l] = \bot$ then
  $T[w, l] \leftarrow\$ \{0,1\}^l$; $(u, x) \leftarrow w$
  If $((u = xk)$ and $(l = \ell))$ then
    If $(S[w] = \bot)$ then $S[w] \leftarrow\$ \{0,1\}^\ell$; $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{T[w, l] \leftarrow S[w]}$
Return $T[w, l]$

Game G_2 / $\boxed{\mathrm{G}_3}$

$St \leftarrow \varepsilon$; $xk \leftarrow\$ \{0,1\}^k$ $S \leftarrow \emptyset$
$c' \leftarrow\$ \mathcal{G}^{\mathrm{RoR},\mathrm{EXPOSE},\mathrm{RO}}(xk)$; Return $(c' = 1)$

$\mathrm{RoR}(m, \eta)$

If $s = 1$ then return $\bot$
$(n, St) \leftarrow\$ \mathsf{NG}(\eta, St)$
$r \leftarrow R[m, n]$; $R[m, n] \leftarrow\$ \{0,1\}^\ell$
If $((m, n) \in S)$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{R[m, n] \leftarrow r}$
If $(T[(xk, (m, n)), \ell] \neq \bot)$ then
  $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{R[m, n] \leftarrow T[(xk, (m, n)), \ell]}$
$S \leftarrow S \cup \{(m, n)\}$
Return $R[m, n]$

EXPOSE

$s \leftarrow 1$
Return St

$\mathrm{RO}(w, l)$

If $(T[w, l] = \bot)$ then
  $T[w, l] \leftarrow\$ \{0,1\}^l$; $(u, x) \leftarrow w$; $(m, n) \leftarrow x$
  If $((u = xk)$ and $(l = \ell)$ and $((m, n) \in S))$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{T[w, l] \leftarrow R[m, n]}$
Return $T[w, l]$

Adversary $\mathcal{P}^{\mathrm{GEN}}$

$xk \leftarrow\$ \{0,1\}^k$; $i \leftarrow 0$
$g \leftarrow\$ \{1, \ldots, q_2\}$
$c' \leftarrow\$ \mathcal{G}^{\mathrm{RoR},\mathrm{EXPOSE},\mathrm{RO}}(xk)$
Return n_g

$\mathrm{RoR}(m, \eta)$

If $s = 1$ then return $\bot$
GEN(η)
$r \leftarrow\$ \{0,1\}^\ell$
Return r

EXPOSE

$s \leftarrow 1$
$St \leftarrow$ EXPOSE
Return St

$\mathrm{RO}(w, l)$

If $(T[w, l] = \bot)$ then
  $T[w, l] \leftarrow\$ \{0,1\}^l$
  $(u, x) \leftarrow w$; $(m, n) \leftarrow x$
  $i \leftarrow i + 1$; $n_i \leftarrow n$
Return $T[w, l]$

Fig. 3. Games and adversary for proof of Lemma 1.

Equation 7 is a standard re-formulation of the definition of the advantage. In game G_2, replies to queries to the RoR and RO oracles are independently distributed, so that it is equivalent to the $c = 0$ case of game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$, up to

the flipping of the outcomes from true to false. In game G_3, the replies to RoR queries are given by $\mathsf{HE}^{\mathrm{RO}}$, making it equivalent to the $c = 1$ case of game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$. This justifies Eq. 8. Games G_2, G_3 are identical until bad, so the Fundamental Lemma of Game Playing [10] justifies Eq. 9. In game G_2, replies to all oracle queries are random and independent, so adversary $\mathcal{P}$ can simulate the oracles of adversary $\mathcal{G}$ directly. Its output is a random one of the nonces in a RO-query of $\mathcal{G}$, whence Eq. 10. Equation 11 is because $\mathcal{P}$ makes q_1 calls to its GEN oracle and q_3 queries to its EXPOSE oracle. □

Standard-model hedged extractor. Next we give a standard-model hedged extractor based on a PRF F and an almost-XOR-universal hash function H. We use the latter essentially as a strong extractor. The construction is simple: the PRF is evaluated on the message and nonce, and the hash function is evaluated only on the nonce. The results are combined via a simple XOR operation. The intuition behind this is that as long as at least one of the outputs generated by the two schemes is random, then the result is also random. PRF security of the hedged extractor is proved assuming only on the assumed PRF security of F. Real-or-random security of the hedged extractor is shown for a restricted class of nonce generators NG and adversaries $\mathcal{G}$. Namely NG must retain unpredictability even in the presence of an EXPOSE query revealing the state, and $\mathcal{G}$'s RoR queries must not depend on the seed. These restrictions reflect inherent limitations of strong extractors.

We start with some definitions. For $\epsilon \in [0,1]$, function family H is *ϵ-almost XOR-universal* [22] if $\mathsf{H.Rng} = \{0,1\}^\ell$ for some $\ell \in \mathbb{N}$ and, for all distinct $x, y \in \mathsf{H.Dom}$ and all $s \in \{0,1\}^\ell$, we have

$$\Pr[\mathsf{H}(hk, x) \oplus \mathsf{H}(hk, y) = s \ : \ hk \leftarrow\!\!\$\ \mathsf{H.Keys}] \leq \epsilon.$$

Our standard model construction is as follows. Let $\mathsf{F}\colon\ \mathsf{F.Keys} \times (\{0,1\}^* \times \{0,1\}^*) \to \{0,1\}^\ell$ be a function family assumed to be a PRF, and let $\mathsf{H}\colon\ \mathsf{H.Keys} \times \mathsf{H.Dom} \to \{0,1\}^\ell$ be an almost-XOR-universal hash function with $\mathsf{H.Dom} \subseteq \{0,1\}^*$. We associate to $\ell, \mathsf{F}, \mathsf{H}$ the standard-model hedged extractor $\mathsf{HE} = \mathbf{HE2}[\mathsf{F}, \mathsf{H}]\colon\ (\mathsf{F.Keys} \times \mathsf{H.Keys}) \times (\{0,1\}^* \times \mathsf{H.Dom}) \to \{0,1\}^\ell$ defined by

$$\frac{\text{Algorithm } \mathsf{HE}(xk, (x, y))}{(hk, fk) \leftarrow xk\ ;\ z_1 \leftarrow \mathsf{H}(hk, y)\ ;\ z_2 \leftarrow \mathsf{F}(fk, (x, y))\ ;\ \text{Return } z_1 \oplus z_2}$$

Here $\mathsf{HE.Keys} = \mathsf{F.Keys} \times \mathsf{H.Keys}$, $\mathsf{HE.Dom} = \{0,1\}^* \times \mathsf{HE.NS}$ with $\mathsf{HE.NS} = \mathsf{H.Dom}$, and $\mathsf{HE.Rng} = \{0,1\}^\ell$. The following says this hedged extractor achieves PRF security and restricted real-or-random security.

Lemma 2. *Let $\ell \geq 1$ be an integer. Let $\mathsf{F}\colon\ \mathsf{F.Keys} \times (\{0,1\}^* \times \{0,1\}^*) \to \{0,1\}^\ell$ be a function family. Let $\mathsf{H}\colon\ \mathsf{H.Keys} \times \mathsf{H.Dom} \to \{0,1\}^\ell$ be a $(1+\gamma) \cdot 2^{-\ell}$-almost-XOR-universal hash function. Let $\mathsf{HE} = \mathbf{HE2}[\mathsf{F}, \mathsf{H}]$ be the function family associated to ℓ, F and H as above.*

1. *If $\mathcal{A}$ is an adversary making q queries to its* Fn *oracle then there is an adversary $\mathcal{B}$ (described in the proof) such that*

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{A}) \le \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(\mathcal{B}) \ . \tag{12}$$

Adversary $\mathcal{B}$ also makes q queries to its Fn *oracle and has running time that of $\mathcal{A}$ plus the time for q computations of* H.

2. *Let* NG *be a nonce generator that produces outputs in the set* $\mathsf{H.Dom}$. *If $\mathcal{G}$ is an agnostic adversary making q queries to its* RoR *oracle and* $\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(q,1) \le 2^{-m}$, *then*

$$\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) \le \frac{q}{2}\sqrt{\gamma + \frac{2^{\ell}}{2^m}} \ . \tag{13}$$

To prove this we first need some more definitions. Recall that for $\epsilon \in [0,1]$, a function family H is *ϵ-almost universal* if $\Pr[\mathsf{H}(hk,x) = \mathsf{H}(hk,y) \ : \ hk \leftarrow\!\!\$\ \mathsf{H.Keys}] \le \epsilon$ for all distinct $x, y \in \mathsf{H.Dom}$. For a function family H with $\mathsf{H.Dom} = \{0,1\}^* \times \{0,1\}^*$, we say that H is *ϵ-almost universal in the second input component* if for all $x_1, y_1 \in \{0,1\}^*$ and all $x_2, y_2 \in \{0,1\}^*$ with $x_2 \ne y_2$, $\Pr[\mathsf{H}(hk,(x_1,x_2)) = \mathsf{H}(hk,(y_1,y_2)) \ : \ hk \leftarrow\!\!\$\ \mathsf{H.Keys}] \le \epsilon$. For $k \in \mathbb{N}$ and $\epsilon \in [0,1]$, a (k,ϵ)-strong extractor SE is a (standard model) family of functions such that $\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{E},\mathsf{NG}}(1,0,1) \le \epsilon$ for all NG with $\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(1,1) \le 2^{-k}$. This is a re-formulation of the standard requirement that is clearly equivalent to it. The Leftover Hash Lemma, a celebrated result by Impagliazzo, Levin, and Luby [20], states that an almost universal hash function is a strong extractor:

Lemma 3 (Leftover Hash Lemma). *Let $\gamma, k > 0$. Let H be an $(1+\gamma)/|\mathsf{H.Rng}|$-almost universal hash function family. Then H is an (k,ϵ)-strong extractor where*

$$\epsilon = \frac{1}{2}\sqrt{\gamma + \frac{|\mathsf{H.Rng}|}{2^k}} \ .$$

The original result is stated in a different formalism and a slightly more restricted form, but it generalizes to the form stated here. (The formulation of the bounds comes from [2].)

Since we use the nonce that is used in the hash function H also outside of it, namely in the evaluation of the PRF F, we cannot immediately apply the leftover-hash lemma. However, we show that if the function H is almost XOR-universal, then the hedged extractor HE obtained by XORing the output of F to the output of H is almost universal and therefore serves as a strong extractor.

Lemma 4. *Let* $\mathsf{H} : \mathsf{H.Keys} \times \mathsf{H.Dom} \to \{0,1\}^{\ell}$ *be an ϵ-almost-XOR-universal hash function. Let* $\mathsf{F} : \mathsf{F.Keys} \times (\{0,1\}^* \times \{0,1\}^*) \to \{0,1\}^{\ell}$ *a function family and* $\mathsf{H.Dom} \subseteq \{0,1\}^*$. *Then* HE *as defined above is ϵ-almost-universal in the second input component.*

Proof. For all $m, m' \in \{0,1\}^*$ and $n, n' \in \mathsf{H.Dom}$ with $n \neq n'$:

$$\begin{aligned}
&\Pr\left[\mathsf{HE}(xk,(m,n)) = \mathsf{HE}(xk,(m',n')) \;:\; xk \leftarrow\!\!\$\; \mathsf{HE.Keys}\right] \\
&\quad = \Pr\Big[\mathsf{H}(hk,n) \oplus \mathsf{F}(fk,(m,n)) = \mathsf{H}(hk,n') \oplus \mathsf{F}(fk,(m',n')) \\
&\qquad\qquad : \; (hk,fk) \leftarrow\!\!\$\; \mathsf{H.Keys} \times \mathsf{F.Keys}\Big] \\
&\quad \leq \max_{fk \in \mathsf{F.Keys}} \Pr\Big[\mathsf{H}(hk,n) \oplus \mathsf{H}(hk,n') = \mathsf{F}(fk,(m,n)) \oplus \mathsf{F}(fk,(m',n')) \\
&\qquad\qquad : \; hk \leftarrow\!\!\$\; \mathsf{H.Keys}\Big] \leq \epsilon,
\end{aligned}$$

since for each $fk \in \mathsf{F.Keys}$ the term $\mathsf{F}(fk,(m,n)) \oplus \mathsf{F}(fk,(m',n'))$ describes a fixed value $s \in \{0,1\}^\ell$ and H is ϵ-almost XOR-universal. □

A strong extractor will only guarantee the outputs to be random and independent of the seed if the inputs to the extractor, that is the nonces, do not depend on the seed. Once the seed is exposed to the adversary, no guarantee on further outputs can be given. Therefore, for the game $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$ we restrict our attention to agnostic adversaries $\mathcal{G}$.

Proof (Lemma 2). We first prove Eq. 12. Adversary $\mathcal{B}$ starts by choosing a seed $hk \leftarrow\!\!\$\; \mathsf{H.Keys}$ uniformly at random. It then uses the assumed adversary $\mathcal{A}$, and whenever $\mathcal{A}$ makes a query $\textsc{Fn}(x,y)$, then $\mathcal{B}$ first computes $z_1 \leftarrow \mathsf{H}(hk,y)$ and then makes a query $z_2 \leftarrow \textsc{Fn}(x,y)$, and returns $z_1 \oplus z_2$. Finally, $\mathcal{B}$ provides the same output as $\mathcal{A}$. The view of $\mathcal{A}$ has the same distribution in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{A})$ and in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{B})$, and therefore the distribution of the output is also the same. This implies Eq. 12.

We prove Eq. 13 using a hybrid argument similarly to Zuckerman [33, Lemma 6]. The hybrid argument involves a game G as specified in Fig. 4 and adversaries $\mathcal{G}_1, \ldots, \mathcal{G}_q$ that each query the oracle RoR only once. This is achieved by having $\mathcal{G}_i$ answer the ith query of $\mathcal{G}$ by using the RoR oracle, by using for all previous queries uniformly random values and computing all subsequent values by evaluating HE. In more detail, for all queries $j = 1, \ldots, i-1$, the adversary $\mathcal{G}_i$ will call its $\textsc{Gen}(\eta)$ oracle and then sample a value from $\mathsf{HE.Rng} = \{0,1\}^\ell$ uniformly at random. For the ith query, the adversary $\mathcal{G}_i$ will call $\textsc{RoR}(m,\eta)$ and obtain the output r, as well as the nonce generator's state St through a subsequent EXPOSE query. The answer to $\mathcal{G}$ will be r in this case. For all queries $j = i+1, \ldots, q$, adversary $\mathcal{G}_i$ will compute the nonce via $(n_j, St) \leftarrow \mathsf{NG}(\eta, St)$ and the output as $\mathsf{HE}(xk, m, n_j)$. This is possible because it uses the seed xk only after the challenge query. We can therefore specify the adversary as a pair $\mathcal{G}_i = (\mathcal{G}_{i,\mathsf{prep}}, \mathcal{G}_{i,\mathsf{chal}})$ as described in Fig. 4.

To conclude the hybrid argument, we first observe that the view of $\mathcal{G}$ in $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$ with $c = 1$ is the same as its view in $\mathrm{G}(\mathcal{G}_1)$ with $c = 1$. Subsequently, for each $i = 1, \ldots, q-1$, the view of $\mathcal{G}$ in $\mathrm{G}(\mathcal{G}_i)$ with $c = 0$ is the same as in $\mathrm{G}(\mathcal{G}_{i+1})$ with $c = 1$. Finally, the view of $\mathcal{G}$ in $\mathrm{G}(\mathcal{G}_q)$ with $c = 0$ is the same as in $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G})$ with $c = 0$.

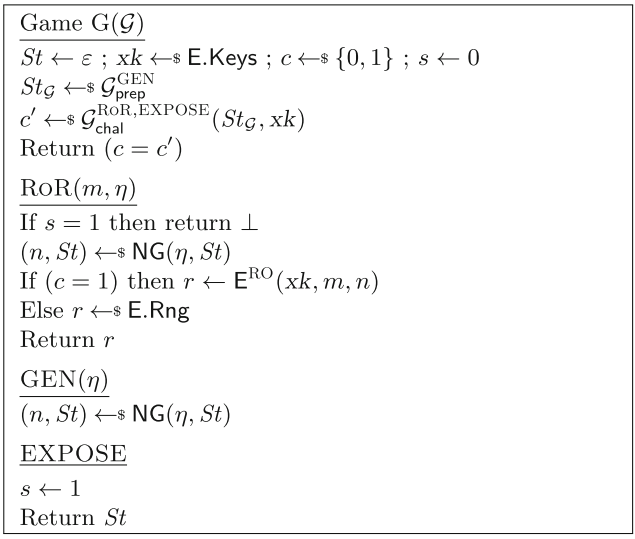

Fig. 4. Hybrid game for E, as needed in the proof of Lemma 2.

We still have to bound the advantage of adversary $\mathcal{G}_i$. By Lemma 4, we know that HE is $(1+\gamma)/2^{\ell}$-almost universal in its second input and therefore by a slight generalization of Lemma 3 an (m, ϵ)-strong extractor for $\epsilon = 1/2\sqrt{\gamma + 2^{\ell-m}}$. Fix i, then $\mathcal{G}_{i,\mathsf{prep}}$ makes $i-1$ queries to its GEN oracle. We can therefore define a nonce generator $\mathsf{RG}(\mathcal{G}_{i,\mathsf{guess}}, \mathsf{NG})$ as follows: Set $St \leftarrow \varepsilon$ and execute $(n, St_{\mathcal{G}}) \leftarrow\$ \mathcal{G}^{\mathrm{GEN}}_{i,\mathsf{guess}}$ with the oracle as defined in the game. Compute a random x via $(x, St) \leftarrow\$ \mathsf{NG}(\eta, St)$ and output a pair $(x, (St, St_{\mathcal{G}}))$.

This implies that $\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{RG}(\mathcal{G}_{i,\mathsf{prep}},\mathsf{NG})}(1,1) \leq \mathbf{Adv}^{\mathsf{pred}}_{\mathsf{NG}}(i,1)$, since otherwise $\mathcal{G}_{i,\mathsf{prep}}$ could be used in the statement against the assumption on NG. But now we can view $\mathcal{G}_{i,\mathsf{chal}}$ as an adversary against HE in the original $\mathbf{G}^{\mathrm{ror}}$ game, and $2\Pr[\mathrm{HYB}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}_i)] - 1 \leq \mathbf{Adv}^{\mathsf{ror}}_{\mathsf{SE},\mathsf{RG}(\mathcal{G}_{i,\mathsf{prep}},\mathsf{NG})}(\mathcal{G}_{i,\mathsf{chal}}) \leq \epsilon$. We obtain a factor q through the hybrid argument; this completes the proof. □

4 Nonce-Based Public-Key Encryption

In this section we define nonce-based public-key encryption, giving first a syntax and then two security goals, NBP1 and NPB2. We give a construction, simultaneously meeting both goals, based on a hedged extractor. Instantiating the latter via our constructions of Sect. 3 yields concrete nonce-based PKE schemes, one in the ROM and the other in the standard model.

Syntax. A *nonce-based public-key encryption scheme* NPE specifies the following. Receiver key-generation algorithm $\mathsf{NPE.Kg}$ returns an encryption key ek and associated decryption key dk. Sender seed-generation algorithm $\mathsf{NPE.sKg}$ returns a seed xk. Encryption algorithm $\mathsf{NPE.Enc}$ takes ek, xk, message $m \in \{0,1\}^*$ and

nonce n from nonce set $\mathsf{NPE.NS}$ to return a ciphertext c. This algorithm is *deterministic*. Decryption algorithm $\mathsf{NPE.Dec}$ (also deterministic) takes ek, dk and ciphertext c to return a value in $\{0,1\}^* \cup \{\perp\}$. The scheme NPE is correct if for all $(ek, dk) \in [\mathsf{NPE.Kg}]$, all $xk \in [\mathsf{NPE.sKg}]$, all $m \in \{0,1\}^*$ and all $n \in \mathsf{NPE.NS}$ we have $\mathsf{NPE.Dec}(ek, dk, \mathsf{NPE.Enc}(ek, xk, m, n)) = m$.

The receiver key-generation algorithm has the same role as in a randomized PKE scheme. The sender seed-generation algorithm is a new element of nonce-based PKE schemes. Encryption is changed to take a nonce rather than randomness and, importantly, is now deterministic. Decryption is as in a standard PKE scheme. Unlike symmetric nonce-based encryption, the decryption algorithm is not given the nonce.

Nonce-based encryption is a sender-side hardening and can be added to an existing encryption scheme in such a way that the receiver is oblivious to its presence and the receiver implementation needs no changes.

Game $\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE,NG}}(\mathcal{A})$	Game $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE,NG}}(\mathcal{A})$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{NPE.Kg}$; $xk \leftarrow\!\!\$\ \mathsf{NPE.sKg}$	$(ek, dk) \leftarrow\!\!\$\ \mathsf{NPE.Kg}$
$b \leftarrow\!\!\$\ \{0,1\}$; $St \leftarrow \varepsilon$; $C, S_0, S_1 \leftarrow \emptyset$	$xk \leftarrow\!\!\$\ \mathsf{NPE.sKg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{ENC,DEC,RO}}(ek)$	$b \leftarrow\!\!\$\ \{0,1\}$; $St \leftarrow \varepsilon$; $C \leftarrow \emptyset$
Return $(b = b')$	$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{ENC,DEC,RO}}(ek, xk)$
$\underline{\mathrm{ENC}(m_0, m_1, \eta)}$	Return $(b = b')$
If $(\lvert m_0\rvert \neq \lvert m_1\rvert)$ then return $\perp$	$\underline{\mathrm{ENC}(m_0, m_1, \eta)}$
$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$	If $(\lvert m_0\rvert \neq \lvert m_1\rvert)$ then return $\perp$
If $((m_0, n) \in S_0)$ then return $\perp$	$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$
If $((m_1, n) \in S_1)$ then return $\perp$	$c \leftarrow \mathsf{NPE.Enc}^{\mathrm{RO}}(ek, xk, m_b, n)$
$S_0 \leftarrow S_0 \cup \{(m_0, n)\}$	$C \leftarrow C \cup \{c\}$
$S_1 \leftarrow S_1 \cup \{(m_1, n)\}$	Return c
$c \leftarrow \mathsf{NPE.Enc}^{\mathrm{RO}}(ek, xk, m_b, n)$	$\underline{\mathrm{DEC}(c)}$
$C \leftarrow C \cup \{c\}$	If $(c \in C)$ then return $\perp$
Return c	$m \leftarrow \mathsf{NPE.Dec}^{\mathrm{RO}}(ek, dk, c)$
$\underline{\mathrm{DEC}(c)}$	Return m
If $(c \in C)$ then $m \leftarrow \perp$	$\underline{\mathrm{RO}(x, l)}$
Else $m \leftarrow \mathsf{NPE.Dec}^{\mathrm{RO}}(ek, dk, c)$	If $T[x, l] = \perp$ then $T[x, l] \leftarrow\!\!\$\ \{0,1\}^l$
Return m	Return $T[x, l]$
$\underline{\mathrm{RO}(x, l)}$	
If $T[x, l] = \perp$ then $T[x, l] \leftarrow\!\!\$\ \{0,1\}^l$	
Return $T[x, l]$	

Fig. 5. Games for security goals for nonce-based asymmetric encryption scheme NPE. Both are relative to nonce generator NG.

Security definitions. Let NPE be a nonce-based PKE scheme. Let NG be a nonce generator returning nonces in NPE.NS. We associate to NPE, NG and adversary $\mathcal{A}$ the games of Fig. 5. We define the advantages of $\mathcal{A}$ in these games via

$$\mathbf{Adv}^{\mathsf{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) = 2\Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})] - 1$$

and

$$\mathbf{Adv}^{\mathsf{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) = 2\Pr[\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})] - 1\ ,$$

respectively. The games are described in the ROM; standard-model definitions are derived by considering only schemes and adversaries that do not query the RO oracle. The notation "nbp" stands for "nonce-based privacy." We proceed to discuss the definitions.

Game $\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ formalizes security in the case where the sender's seed is not exposed, captured in the formalism by the fact that the adversary is not given the seed as input, while game $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ formalizes security in the case where the sender's seed is exposed, formalized by its being given to the adversary as input. Both ask for indistinguishability-style security under a chosen-ciphertext attack. In $\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$, the natural restriction that one would consider is to ask that nonces not repeat. The restriction we make is weaker, resulting in a stronger security condition, namely that message-nonce pairs may not repeat. (Thus, security is provided even if a nonce repeats, as long as the message is different.) We will achieve this notion for *any* nonce generator, meaning we get very good privacy with minimal restrictions on nonces. In $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$, no restriction is made, so a priori the notion is stronger, but we will achieve it only if the hedged extractor is ror-secure, which will be further reduced to unpredictability of the nonce generator. Thus, in this case, security requires unpredictable nonces.

In game $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$, we say that adversary $\mathcal{A}$ is *agnostic* if its ENC, DEC queries do not depend on the seed xk. More formally, there exists a pair $(\mathcal{A}_1, \mathcal{A}_2)$ of algorithms such that $\mathcal{A}^{\mathrm{ENC,DEC,RO}}(ek, xk)$ does the following:

$$St \leftarrow\!\!\$\ \mathcal{A}_1^{\mathrm{ENC,DEC,RO}}(ek)\ ;\ c' \leftarrow \mathcal{A}_2^{\mathrm{RO}}(xk, St)\ ;\ \ \text{Return } c'\ .$$

Scheme. We specify a scheme NPE that achieves both the NPB1 and NPB2 security notions simultaneously, that is, it guarantees security if either the sender's state remains secret and as long as the message-nonce pairs are unique, or even if the sender's state is leaked to the adversary as long as the nonces are sufficiently unpredictable. The construction is actually a transform **R2NPE** that takes a base, randomized PKE scheme PE and a hedged extractor HE to return a nonce-based PKE scheme $\mathsf{NPE} = \mathbf{R2NPE}[\mathsf{PE}, \mathsf{HE}]$ whose algorithms are described in Fig. 6. The nonce space is that of the hedged extractor, i.e. $\mathsf{NPE.NS} = \mathsf{HE.NS}$. The construction requires that the randomness provided by HE is sufficient, i.e., $\mathsf{PE.rl} = \mathsf{HE.ol}$.

We first prove that the scheme achieves NBP1-security, that is, it is secure as a nonce-based scheme as long as the sender's seed remains secret. The theorem bounds the adversaries advantage by advantages of other, related adversaries against the underlying probabilistic public-key scheme and the PRF-property of the hedged extractor HE. For the constructions described in Sect. 3, the latter

Algorithm $\mathsf{NPE.Kg}$	Algorithm $\mathsf{NPE.Enc}^{\mathrm{RO}}(ek, xk, m, n)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PE.Kg}$	$r \leftarrow \mathsf{HE}^{\mathrm{RO}}(xk, (m, n))$
Return (ek, dk)	$c \leftarrow \mathsf{PE.Enc}(ek, m; r)$
	Return c
Algorithm $\mathsf{NPE.sKg}$	Algorithm $\mathsf{NPE.Dec}(ek, dk, c)$
$xk \leftarrow\!\!\$\ \mathsf{HE.Keys}$	$m \leftarrow \mathsf{PE.Dec}(ek, dk, c)$
Return xk	Return m

Fig. 6. The nonce-based public-key encryption scheme NPE, based on probabilistic public-key encryption scheme PE and hedged extractor HE. The nonce space is the same as for the hedged extractor.

advantage is then bounded by Lemmas 1 (for the ROM construction) and 2 (for the standard-model construction), respectively.

Theorem 5. *Let* PE *be a (standard, randomized) public-key encryption scheme. Let* HE *be a hedged extractor. Let nonce-based public-key encryption scheme* $\mathsf{NPE} = \mathbf{R2NPE}[\mathsf{PE}, \mathsf{HE}]$ *be associated to them as above. Let* NG *be a nonce generator. Let* $\mathcal{A}$ *be an adversary making at most* q_1 *queries to its* ENC *oracle,* q_2 *queries to its* DEC *oracle, and* q_3 *queries to its* RO *oracle. Then the proof specifies adversaries* $\mathcal{B}$ *and* $\mathcal{G}$ *such that*

$$\mathbf{Adv}^{\mathsf{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{G}) + \mathbf{Adv}^{\mathsf{ind}}_{\mathsf{PE}}(\mathcal{B}) \;, \tag{14}$$

where adversary $\mathcal{B}$ *makes at most* q_1 *queries to its* ENC *oracle and* q_2 *queries to its* DEC *oracle; in terms of computation it evaluates* NG *for* q_1 *times and manages an array of generated ciphertexts. Adversary* $\mathcal{G}$ *makes at most* q_1 *queries to its* FN *oracle and* q_3 *queries to its* RO *oracle; it generates keys for* PE *and compute* q_1 *encryptions and* q_2 *decryptions in addition to the computation of* $\mathcal{A}$*.*

Proof. Adversary $\mathcal{B}$ obtains ek and starts by setting $St \leftarrow \varepsilon$, $S_0 \leftarrow \emptyset$, and $S_1 \leftarrow \emptyset$. Adversary $\mathcal{B}$ then runs the adversary $\mathcal{A}$ internally on input ek. For queries $\mathrm{ENC}(m_0, m_1, \eta)$, adversary $\mathcal{B}$ computes $(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$, checks whether $(m_0, n) \in S_0$ or $(m_1, n) \in S_1$, and returns $\perp$ in that case. Otherwise, it queries its own oracle $\mathrm{ENC}(m_0, m_1)$ and returns the result to $\mathcal{A}$. Queries $\mathrm{DEC}(c)$ are answered using the respective oracles in the game, and potential queries $\mathrm{RO}(x, l)$ are answered by emulating a random oracle via lazy sampling as also described in the game.

For each $d \in \{0, 1\}$, adversary $\mathcal{G}_d$ against the PRF initializes $St \leftarrow \varepsilon$, $S_0 \leftarrow \emptyset$, and $S_1 \leftarrow \emptyset$ as $\mathcal{B}$ and generates a key pair $(ek, dk) \leftarrow\!\!\$\ \mathsf{PE.Kg}$. Adversary $\mathcal{G}_d$ then runs adversary $\mathcal{A}$ on input ek. Upon a query $\mathrm{ENC}(m_0, m_1, \eta)$ from $\mathcal{A}$, adversary $\mathcal{G}_d$ computes $(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$, checks whether $(m_0, n) \in S_0$ or $(m_1, nonce) \in S_1$, and returns $\perp$ in that case. $\mathcal{G}_d$ gets r from its oracle $\mathrm{FN}(m_d, n)$, computes $c \leftarrow \mathsf{PE.Enc}(ek, m_d; r)$ and returns c. Queries $\mathrm{DEC}(c)$ are answered by computing $m \leftarrow \mathsf{PE.Dec}(ek, dk, c)$ and returning m. Potential queries $\mathrm{RO}(x, l)$

are referred to the random oracle provided in the game. When $\mathcal{A}$ provides its output b', then $\mathcal{G}_d$ outputs $b' \oplus d$.

We define a hybrid game G in which all computations are performed as before, except that in the encryption with a new pair of message m_b and nonce n a fresh random string $r \leftarrow \{0,1\}^\ell$ is used. The difference with the game $\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ can be highlighted as described in game G in Fig. 7.

Game G	Game H
$\vdots$	$\vdots$
$S_1 \leftarrow S_1 \cup \{(m_1, n)\}$	$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$
$c \leftarrow \mathsf{NPE.Enc}^{\mathrm{RO}}(ek, xk, m_b, n)$	$c \leftarrow \mathsf{NPE.Enc}^{\mathrm{RO}}(ek, xk, m_b, n)$
$r \leftarrow\!\!\$\ \{0,1\}^\ell$	$r \leftarrow\!\!\$\ \{0,1\}^\ell$
$c \leftarrow \mathsf{PE.Enc}(ek, m_b; r)$	$c \leftarrow \mathsf{PE.Enc}(ek, m_b; r)$
$C \leftarrow C \cup \{c\}$	$C \leftarrow C \cup \{c\}$
Return c	Return c
$\vdots$	$\vdots$

Fig. 7. Intermediate games used in the proofs of Theorem 5 (left) and Theorem 6 (right).

The view of $\mathcal{A}$ in G is exactly the same as in $\mathbf{G}^{\mathrm{ind}}_{\mathsf{PE}}(\mathcal{B})$. Furthermore, we observe that

$$2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_0)] \;=\; \Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) | b = 0] - \Pr[\mathrm{G} | b = 0] + 1,$$

the reason is that if the bit $b = 0$ is chosen in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_0)$, then the view of $\mathcal{A}$ is exactly as in G with $b = 0$; all ciphertexts are encryptions of m_0 with fresh randomness. Analogously, if $b = 1$ is chosen in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_0)$, then the view of $\mathcal{A}$ is exactly as in $\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ with $b = 0$; all ciphertexts are encryptions of m_0 with randomness computed via NG and HE from the message m_0 and the input η, but in this case $\mathcal{G}_0$ outputs the "wrong" bit. In the same sense,

$$2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_1)] \;=\; \Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) | b = 1] - \Pr[\mathrm{G} | b = 1] + 1,$$

since if the bit b is chosen as $b = 0$ in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_0)$, then the view of $\mathcal{A}$ is exactly as in G with $b = 1$; all ciphertexts are encryptions of m_1 with fresh randomness. This is the reason for $\mathcal{G}_1$ to invert the output of $\mathcal{A}$. Overall, we obtain

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) &= \Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) | b = 0] + \Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) | b = 1] - 1 \\
&= \Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) | b = 0] - \Pr[\mathrm{G} | b = 0] \\
&\quad + \Pr[\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) | b = 1] - \Pr[\mathrm{G} | b = 1] + 2\Pr[\mathrm{G}] - 1 \\
&\leq\; 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_0)] - 1 + 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}_1)] - 1 + \mathbf{Adv}^{\mathsf{nbp1}}_{\mathsf{PE}}(\mathcal{B}) \\
&= \;\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{G}_0) + \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{G}_1) + \mathbf{Adv}^{\mathsf{nbp1}}_{\mathsf{PE}}(\mathcal{B}).
\end{aligned}$$

The proof concludes by defining $\mathcal{G}$ as choosing $\mathcal{G}_0$ or $\mathcal{G}_2$ with probability 1/2 each. □

The property of a hedged extractor to serve simultaneously as extractor and as a PRF implies that the scheme described above is also secure even if the sender's seed leaks, as long as the nonces are sufficiently unpredictable. The reduction in the theorem below preserves agnosticity, meaning if $\mathcal{A}$ is agnostic, so is $\mathcal{G}$. This allows us to draw conclusions based on Lemma 2 in the case of the standard-model hedged extractor.

Theorem 6. *Let* PE *be a (standard, randomized) public-key encryption scheme. Let* HE *be a hedged extractor. Let nonce-based public-key encryption scheme* $\mathsf{NPE} = \mathbf{R2NPE}[\mathsf{PE}, \mathsf{HE}]$ *be associated to them as above. Let* NG *be a nonce generator. Let* $\mathcal{A}$ *be an adversary making at most* q_1 *queries to its* ENC *oracle,* q_2 *queries to its* DEC *oracle, and* q_3 *queries to its* RO *oracle. Then the proof specifies adversaries* $\mathcal{B}$ *and* $\mathcal{G}$ *such that*

$$\mathbf{Adv}^{\mathsf{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A}) \le 2\mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) + \mathbf{Adv}^{\mathsf{ind}}_{\mathsf{PE}}(\mathcal{B}) \;, \tag{15}$$

where adversary $\mathcal{B}$ *makes at most* q_1 *queries to its* ENC *oracle and* q_2 *queries to its* DEC *oracle, and emulates a random oracle for* q_3 *queries. Adversary* $\mathcal{G}$ *makes at most* q_1 *queries to its* RoR *oracle and* q_3 *queries to its* RO *oracle; it generates keys for* PE *and computes* q_1 *encryptions and* q_2 *decryptions in addition to the computation of* $\mathcal{A}$*. Furthermore, if* $\mathcal{A}$ *is agnostic, then* $\mathcal{G}$ *is also agnostic.*

Proof. The proof follows the same ideas as the one for Theorem 5. Adversary $\mathcal{B}$ against the underlying public-key encryption scheme PE behaves as follows. It obtains an encryption key ek as an input and generates a seed $xk \leftarrow\!\!\$\ \mathsf{HE.Keys}$, and then runs adversary $\mathcal{A}$ on input (ek, xk). Upon a query $\mathrm{ENC}(m_0, m_1, \eta)$ by $\mathcal{A}$, adversary $\mathcal{B}$ queries its oracle $\mathrm{ENC}(m_0, m_1)$ and returns the result to $\mathcal{A}$. Queries $\mathrm{DEC}(c)$ are answered by making the same query to its own oracle; the queries $\mathrm{RO}(x, l)$ are answered by emulating a random oracle via lazy sampling. Adversary $\mathcal{B}$ outputs the same output bit as $\mathcal{A}$.

For each $d \in \{0, 1\}$, adversary $\mathcal{G}_d$ against the extractor behaves as follows. It obtains an extractor seed $xk \in \mathsf{HE.Keys}$, generates a key pair $(ek, dk) \leftarrow\!\!\$\ \mathsf{PE.Kg}$, and then runs adversary $\mathcal{A}$ on input (ek, xk). Upon a query $\mathrm{ENC}(m_0, m_1, \eta)$ from $\mathcal{A}$, adversary $\mathcal{G}_d$ calls $\mathrm{RoR}(m_d, \eta)$ to obtain a random string r. It then computes $c \leftarrow \mathsf{PE.Enc}(ek, m_d; r)$ and returns c. Queries $\mathrm{DEC}(c)$ are answered by computing $m \leftarrow \mathsf{PE.Dec}(ek, dk, c)$ and returning m. Potential queries $\mathrm{RO}(x, l)$ are referred to the random oracle provided in the game. When $\mathcal{A}$ provides its output b', then $\mathcal{G}_d$ outputs $b' \oplus d$. Note that $\mathcal{G}_d$ uses the seed only through $\mathcal{A}$, and if $\mathcal{A}$ uses it only after all encryption queries, then $\mathcal{G}_d$ uses it only after all RoR queries. Thus, if $\mathcal{A}$ is agnostic, then $\mathcal{G}_d$ is also agnostic.

We define a hybrid game H in which all computations are performed as before, except that in the encryption a fresh random string $r \leftarrow \{0, 1\}^\ell$ is used; the difference with the game $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ is highlighted by the boxed code in Fig. 7.

The view of $\mathcal{A}$ in H is exactly the same as in $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{PE}}(\mathcal{B})$. Furthermore, we observe that

$$2\Pr[\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}_0)] \;=\; \Pr[\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})|b=0] - \Pr[\mathrm{H}|b=0] + 1,$$

the reason is that if the bit $b = 0$ is chosen in $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}_0)$, then the view of $\mathcal{A}$ is exactly as in H with $b = 0$; all ciphertexts are encryptions of m_0 with fresh randomness. Analogously, if $b = 1$ is chosen in $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}_0)$, then the view of $\mathcal{A}$ is exactly as in $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ with $b = 0$; all ciphertexts are encryptions of m_0 with randomness computed via HE and NG from the message m_0 and the input η, but in this case $\mathcal{G}_0$ outputs the "wrong" bit. In the same sense,

$$2\Pr[\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}_1)] \;=\; \Pr[\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})|b=1] - \Pr[\mathrm{H}|b=1] + 1,$$

since if the bit b is chosen as $b = 0$ in $\mathbf{G}^{\mathrm{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}_0)$, then the view of $\mathcal{A}$ is exactly as in H with $b = 1$; all ciphertexts are encryptions of m_1 with fresh randomness. This is the reason for $\mathcal{G}_1$ to invert the output of $\mathcal{A}$. The final computation follows exactly as in Theorem 5. □

Settings with multiple senders and multiple receivers. The security properties defined above take into account only a single sender and a single receiver. Realistic settings, however, involve multiple senders and multiple receivers. This means that, on the one hand, encryptions toward the same receivers will be made with respect to different sender seeds. On the other hand, senders will use the same seed to generate randomness for encryptions toward different receivers. To achieve security in these settings, we extend the games $\mathbf{G}^{\mathrm{nbp2}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ and $\mathbf{G}^{\mathrm{nbp1}}_{\mathsf{NPE},\mathsf{NG}}(\mathcal{A})$ by the following oracle:

$$\begin{array}{l}\underline{\mathrm{ENC2}(ek, m, \eta)}\\ (n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)\\ c \leftarrow \mathsf{NPE.Enc}(ek, xk, m, n)\\ \text{Return } c\end{array}$$

The scheme discussed above can easily be shown to achieve the correspondingly modified games; the hedged extractor provides uniform and independent randomness for each encryption.

Extending this game to multiple senders and multiple receives is done by generating multiple sets of keys and seeds in the game and extending the oracles with arguments to select the desired sender and/or receiver of the messages to be processed. The proof then follows by two hybrid arguments, one for reducing the number of senders, and one for reducing the number of receivers. The oracle ENC2 is required in the step to reduce the number of receivers to simulate encryptions toward the receivers not captured in the game.

5 Nonce-Based Signatures

In this section we define and construct nonce-based digital signature schemes.

Background. Eliminating randomness in signing is not new and is easily done. A simple way to convert a given randomized EUF-CMA digital signature scheme DS into a deterministic one is as follows. Let F be a PRF. The key-generation algorithm lets $(sk, vk) \xleftarrow{\$} \mathsf{DS.Kg}$ and $fk \xleftarrow{\$} \mathsf{F.Keys}$, and stores the pair (fk, sk) as the secret signing key of the new scheme. A signature on a message $m \in \{0,1\}^*$ is then computed by first evaluating $r \leftarrow \mathsf{F}(fk, m)$ and then $s \leftarrow \mathsf{DS.Sig}(sk, m; r)$. This method goes back to MNPV [23] and it is easy to show that it works, meaning the constructed, deterministic signature scheme retains the EUF-CMA security of the starting randomized one assuming F is a PRF.

The above solution, however, changes the secret key, which is not always desirable. For example it may be a problem to retrofit deployed schemes with the modification, or if the same signature key is used by multiple applications and the format cannot easily be changed. A folklore solution is to leave the keys unchanged and obtain the coins r via a random oracle applied to the existing secret key sk and the message. In the case that DS is ECDSA, this was proven to work by KM [21]. It was proven to work in general (meaning, for any base EUF-CMA scheme) by BPS [9]. Such de-randomization is used in the Ed25519 signature scheme [11] and is specified for DSA and ECDSA in an RFC by Pornin [25].

Nonce-based signatures. In our model, the signer has a secret key as well as a seed. Signing uses both these and a nonce, and is deterministic. If the seed (and secret key) are kept private, we get the usual EUF-CMA level of security, regardless of how nonces are generated. So far this is providing the same security as deterministic signature schemes. The added condition is that if the seed is exposed (but the secret key isn't) then we still retain security as long as the nonces are unpredictable.

The secret key and seed are held by the same entity, namely the signer. So one may ask how it could be that the seed is exposed but the secret key isn't. That is, either the system is secure, in which case both are secure, or not, in which case both are exposed. If so, indeed, nonce-based signatures do not provide anything over and above classical deterministic signatures. However, we can imagine settings where the level of security for the secret key and seed are different. For example the secret key may be already stored in hardware, and the seed not. The seed may be stored at a different place than the signature scheme's secret key, and it may be re-generated at any frequency that seems appropriate for the application (ranging from never, at fixed time intervals, at every system reboot, or at every signature operation). Being a signer-only modification of the signature generation, a user may also use different seeds on different machines, or use the modified scheme with the standard probabilistic one.

The scheme we propose is again based on hedged extractors.

Definitions. A *nonce-based signature scheme* NDS specifies the following. Signer key-generation algorithm $\mathsf{NDS.Kg}$ returns a signature key sk, a verification key vk, and a seed xk. *Deterministic* signature algorithm $\mathsf{NDS.sign}$ takes sk, xk, message $m \in \{0,1\}^*$, and nonce n from nonce set $\mathsf{NDS.NS}$, to return a signature $s \in \{0,1\}^{\mathsf{NDS.ol}}$. Deterministic verification algorithm $\mathsf{NDS.vrf}$ takes

Game $\mathbf{G}^{\mathrm{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$ / Game $\boxed{\mathbf{G}^{\mathrm{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})}$	Algorithm $\mathsf{NDS.Kg}$
$(sk, vk, xk) \leftarrow\!\!\$\ \mathsf{NDS.Kg}$	$xk \leftarrow\!\!\$\ \mathsf{HE.Keys}$
$St \leftarrow \varepsilon\ ;\ M \leftarrow \emptyset$	$(sk, vk) \leftarrow\!\!\$\ \mathsf{DS.Kg}$
$(m, s) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{SIG,RO}}(\ vk\ \boxed{, xk}\ )$	Return (sk, vk, xk)
$v \leftarrow \mathsf{NDS.vrf}(vk, m, s)$	
Return $((v = 1) \wedge (m \notin M))$	Algorithm $\mathsf{NDS.sign}^{\mathrm{RO}}(sk, xk, m, n)$
$\underline{\mathrm{SIG}(m, \eta)}$	$r \leftarrow \mathsf{HE}^{\mathrm{RO}}(xk, (m, n))$
$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$	$c \leftarrow \mathsf{DS.Sig}(sk, m; r)$
$s \leftarrow \mathsf{NDS.sign}(sk, xk, m, n)$	Return c
$M \leftarrow M \cup \{m\}$	
Return s	
$\underline{\mathrm{RO}(x, l)}$	
If $T[x, l] = \perp$ then $T[x, l] \leftarrow\!\!\$\ \{0,1\}^l$	
Return $T[x, l]$	

Fig. 8. Games for security goals for nonce-based digital scheme NDS relative to nonce-generator NG, and our nonce-based digital signature scheme.

vk, message $m \in \{0,1\}^*$, and candidate signature $s \in \{0,1\}^{\mathsf{NDS.ol}}$, to return a bit $b \in \{0,1\}$. The scheme NDS is correct if for all $(sk, vk, xk) \in [\mathsf{NDS.Kg}]$, all $m \in \{0,1\}^*$ and $n \in \mathsf{NDS.NS}$, the verification of true signatures succeeds: $\mathsf{NDS.vrf}(vk, m, \mathsf{NDS.sign}(sk, xk, m, n)) = 1$.

To formalize security, we consider the games $\mathbf{G}^{\mathrm{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$ and $\mathbf{G}^{\mathrm{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$ in Fig. 8 associated to nonce-based signature scheme NDS, nonce generator NG returning nonces in $\mathsf{NDS.NS}$, and adversary $\mathcal{A}$, where the second game includes the boxed code and the first does not. We let

$$\mathbf{Adv}^{\mathsf{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A}) = \Pr[\mathbf{G}^{\mathrm{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})]\ ,\ \text{and}$$
$$\mathbf{Adv}^{\mathsf{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A}) = \Pr[\mathbf{G}^{\mathrm{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})]\ .$$

As usual the games are described in the ROM, with standard-model definitions are derived by considering only schemes and adversaries that do not query the RO oracle. The difference between the games is tiny, and in just one line of the code, namely that in the second game, the adversary gets the seed xk as an additional input. The first game captures the case that the seed is not exposed, and we will guarantee security for any nonce generator. The second game captures the case that the seed is exposed, in which case we will provide security for unpredictable nonce generators.

Regular signature schemes can be viewed as a special case of nonce-based ones where the seed xk is defined to be the empty string and security is measured relative only to the nonce generator that always returns a uniformly random string.

In game $\mathbf{Adv}^{\mathsf{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$, we say that adversary $\mathcal{A}$ is *agnostic* if its SIG queries do not depend on the seed xk. More formally, there exists a pair $(\mathcal{A}_1, \mathcal{A}_2)$ of algorithms such that $\mathcal{A}^{\mathrm{SIG},\mathrm{RO}}(vk, xk)$ does the following:

$$St \leftarrow\!\!\$\ \mathcal{A}_1^{\mathrm{SIG},\mathrm{RO}}(vk)\ ;\ (m, s) \leftarrow \mathcal{A}_2^{\mathrm{RO}}(\mathrm{x}k, St)\ ;\ \ \text{Return } (m, s)\ .$$

Scheme. We specify a transform **R2NDS** that takes a (standard, randomized) signature scheme DS and a hedged extractor HE and returns the nonce-based signature scheme $\mathsf{NDS} = \mathbf{R2NDS}[\mathsf{DS}, \mathsf{HE}]$ whose algorithms are described in Fig. 8. The nonce space is the same as for the hedged extractor, i.e. $\mathsf{NDS.NS} = \mathsf{HE.NS}$. The length of the signatures is preserved, $\mathsf{NDS.ol} = \mathsf{DS.ol}$. The construction requires that HE provides sufficient randomness, i.e., $\mathsf{DS.rl} = \mathsf{HE.ol}$.

We first show that the described scheme NDS is indeed secure according to the game $\mathbf{G}^{\mathsf{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$, that is, in case the seed is not exposed. In this case and if the hedged extractor is a good pseudo-random function, we achieve the same security guarantees as achieved by the original scheme if proper randomness is used.

Theorem 7. *Let* DS *be a (standard, randomized) digital-signature scheme. Let* HE *be a hedged extractor. Let nonce-based digital signature scheme* $\mathsf{NDS} = \mathbf{R2NDS}[\mathsf{DS}, \mathsf{HE}]$ *be associated to them as above. Let* NG *be a nonce generator. Let* $\mathcal{A}$ *be an adversary making at most* q_1 *queries to its* SIG *oracle and* q_2 *queries to its* RO *oracle. Then the proof specifies adversaries* $\mathcal{B}$ *and* $\mathcal{G}$ *such that*

$$\mathbf{Adv}^{\mathsf{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A}) \le \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{HE}}(\mathcal{G}) + \mathbf{Adv}^{\mathsf{uf}}_{\mathsf{DS}}(\mathcal{B})\ , \tag{16}$$

where adversary $\mathcal{B}$ *makes at most* q_1 *queries to its* SIG *oracle; besides emulating a Random Oracle it performs almost the same computation as* $\mathcal{A}$*. Adversary* $\mathcal{G}$ *makes at most* q_1 *queries to its* RoR *oracle and* q_2 *queries to its* RO *oracle; in terms of computation it generates a key pair for* DS *and computes* q_1 *signatures.*

Proof. Adversary $\mathcal{B}$ against DS behaves as follows. When started with input vk, it executes $\mathcal{A}(vk)$. Upon a query $\mathrm{SIG}(m, \eta)$ from $\mathcal{A}$, adversary $\mathcal{B}$ queries $\mathrm{SIG}(m)$, obtaining a signature s, and returns s to $\mathcal{A}$. Queries to the oracle RO are exactly as in the game, that is, by lazy sampling of a random function.

Adversary $\mathcal{G}$ against the hedged extractor behaves as follows. It generates a key pair $(sk, vk) \leftarrow\!\!\$\ \mathsf{DS.Kg}$, and then runs adversary $\mathcal{A}$ on input vk. Upon a query $\mathrm{SIG}(m, \eta)$ from $\mathcal{A}$, adversary $\mathcal{G}$ calls $\mathrm{Fn}(m, \eta)$ to obtain a random string r. It then computes $s \leftarrow \mathsf{DS.Sig}(sk, m; r)$ and returns s. Potential queries $\mathrm{RO}(x, l)$ are referred to the random oracle provided in the game. When $\mathcal{A}$ provides its output (m, s), then $\mathcal{G}$ outputs the result of $\mathsf{DS.Ver}(vk, m, s) = 1$.

We define a hybrid game G that is defined almost identically with $\mathbf{G}^{\mathsf{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$, with the only difference that the randomness used in SIG queries is uniformly random instead of derived via the hedged extractor. The difference with the game $\mathbf{G}^{\mathsf{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$ is highlighted in Fig. 9.

The view of $\mathcal{A}$ in G is the same as in $\mathbf{G}^{\mathsf{uf}}_{\mathsf{DS}}(\mathcal{B})$; in both cases the signatures are computed with fresh randomness.

Game G	Game H
$\vdots$	$\vdots$
SIG(m, η)	SIG(m, η)
$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$	$(n, St) \leftarrow\!\!\$\ \mathsf{NG}(\eta, St)$
$s \leftarrow \mathsf{NDS.sign}(sk, xk, m, n)$	$s \leftarrow \mathsf{NDS.sign}(sk, xk, m, n)$
$r \leftarrow\!\!\$\ \{0,1\}^{\ell}$	$r \leftarrow\!\!\$\ \{0,1\}^{\ell}$
$s \leftarrow \mathsf{DS.Sig}(sk, m; r)$	$s \leftarrow \mathsf{DS.Sig}(sk, m; r)$
$M \leftarrow M \cup \{m\}$	$M \leftarrow M \cup \{m\}$
Return s	Return s
$\vdots$	$\vdots$

Fig. 9. Intermediate games used in the proofs of Theorem 7 (left) and Theorem 8 (right).

We observe that

$$\begin{aligned} 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})] &= \Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})|b=1] + \Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})|b=0] \\ &= \Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})|b=1] + 1 - \Pr[\neg\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})|b=0] \\ &= \Pr[\mathbf{G}^{\mathrm{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})] + 1 - \Pr[\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}}(\mathcal{B})], \end{aligned}$$

the reason is that if the bit b is chosen as $b = 0$ in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})$, then the view of $\mathcal{A}$ is exactly as in $\mathbf{G}^{\mathrm{uf}}_{\mathsf{DS}}(\mathcal{B})$; all signatures are generated using fresh randomness. Analogously, if b is chosen as $b = 1$ in $\mathbf{G}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G})$, then the view of $\mathcal{A}$ is exactly as in $\mathbf{G}^{\mathrm{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$ with $b = 0$; all signatures are generated with randomness computed via HE and NG from the message m and the input η. For $b = 1$ the probability of $\mathcal{G}$ guessing correctly is the same as the probability of $\mathcal{A}$ forging the signature, but for $b = 0$ the probability of $\mathcal{G}$ guessing correctly is the same as the probability of $\mathcal{A}$ *not* forging a signature. The above equation implies

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HE}}(\mathcal{G}) = \mathbf{Adv}^{\mathrm{nbuf1}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{uf}}_{\mathsf{DS}}(\mathcal{B})$$

and therefore the inequality claimed in the theorem statement. □

The second statement concerns the security in the sense of $\mathbf{G}^{\mathrm{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$, that is, the signatures are indeed secure even if the seed is exposed, as long as the nonces contain a sufficient amount of min-entropy. For the scheme based on our standard-model hedged extractor, we again restrict the statement to agnostic adversaries $\mathcal{A}$. The scheme based on our ROM-based hedged extractor is again secure against all (i.e., not necessarily agnostic) adversaries.

Theorem 8. *Let* DS *be a (standard, randomized) digital-signature scheme. Let* HE *be a hedged extractor. Let nonce-based digital signature scheme* NDS = **R2NDS**[DS, HE] *be associated to them as above. Let* NG *be a nonce generator.*

Let $\mathcal{A}$ be an adversary making at most q_1 queries to its SIG *oracle and q_2 queries to its* RO *oracle. Then the proof specifies adversaries $\mathcal{B}$ and $\mathcal{G}$ such that*

$$\mathbf{Adv}^{\mathsf{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ror}}_{\mathsf{HE},\mathsf{NG}}(\mathcal{G}) + \mathbf{Adv}^{\mathsf{uf}}_{\mathsf{DS}}(\mathcal{B}) \ , \tag{17}$$

where adversary $\mathcal{B}$ makes at most q_1 queries to its SIG *oracle; besides emulating a Random Oracle it performs almost the same computation as $\mathcal{A}$. Adversary $\mathcal{G}$ makes at most q_1 queries to its* RoR *oracle and q_2 queries to its* RO *oracle; in terms of computation it generates a key pair for* DS *and computes q_1 signatures. Furthermore, if $\mathcal{A}$ is agnostic, then $\mathcal{G}$ is also agnostic.*

Proof. Adversary $\mathcal{B}$ against DS behaves as follows. When started with input vk, it samples $xk \leftarrow\!\!\$\ \mathsf{HE.Keys}$ and executes $\mathcal{A}(vk, xk)$. Upon a query $\mathrm{SIG}(m, \eta)$ from $\mathcal{A}$, adversary $\mathcal{B}$ queries $\mathrm{SIG}(m)$, obtaining a signature s, and returns s to $\mathcal{A}$. Queries to the oracle RO are exactly as in the game, that is, by lazy sampling of a random function.

Adversary $\mathcal{G}$ against the hedged extractor behaves as follows. It obtains a seed $xk \in \mathsf{HE.Keys}$ and generates a key pair $(sk, vk) \leftarrow\!\!\$\ \mathsf{DS.Kg}$, and then runs adversary $\mathcal{A}$ on input (vk, xk). Upon a query $\mathrm{SIG}(m, \eta)$ from $\mathcal{A}$, adversary $\mathcal{G}$ calls $\mathrm{RoR}(m, \eta)$ to obtain a random string r. It then computes $s \leftarrow \mathsf{DS.Sig}(sk, m; r)$ and returns s. Potential queries $\mathrm{RO}(x, l)$ are referred to the random oracle provided in the game. When $\mathcal{A}$ provides its output (m, s), then $\mathcal{G}$ outputs the result of $\mathsf{DS.Ver}(vk, m, s) = 1$.

We define a hybrid game H that is defined almost identically with $\mathbf{G}^{\mathsf{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$, with the only difference that the randomness used in SIG queries is uniformly random instead of derived via the hedged extractor. The difference with the game $\mathbf{G}^{\mathsf{nbuf2}}_{\mathsf{NDS},\mathsf{NG}}(\mathcal{A})$ is highlighted in Fig. 9.

The view of $\mathcal{A}$ in H is the same as in $\mathbf{G}^{\mathsf{uf}}_{\mathsf{DS}}(\mathcal{B})$; in both cases the signatures are computed with fresh randomness. The remainder of the proof follows almost exactly as in Theorem 7. □

Acknowledgments. Bellare was supported in part by NSF grants CNS-1526801 and CNS-1228890, ERC Project ERCC FP7/615074 and a gift from Microsoft. Tackmann was supported in part by the Swiss National Science Foundation (SNF) via Fellowship No. P2EZP2_155566 and by NSF grant CNS-1228890. We thank the anonymous reviewers for their helpful and detailed comments.

References

1. OpenSSL predictable random number generator. Debian security advisory, May 2008. http://www.debian.org/security/2008/dsa-1571
2. Barak, B., Dodis, Y., Krawczyk, H., Pereira, O., Pietrzak, K., Standaert, F.-X., Yu, Y.: Leftover hash lemma, revisited. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 1–20. Springer, Heidelberg (2011)
3. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)

4. Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: how to protect against bad randomness. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 232–249. Springer, Heidelberg (2009)
5. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 26–45. Springer, Heidelberg (1998)
6. Bellare, M., Fischlin, M., O'Neill, A., Ristenpart, T.: Deterministic encryption: definitional equivalences and constructions without random oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 360–378. Springer, Heidelberg (2008)
7. Bellare, M., Hoang, V.T.: Resisting randomness subversion: fast deterministic and hedged public-key encryption in the standard model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 627–656. Springer, Heidelberg (2015)
8. Bellare, M., Kohno, T., Shoup, V.: Stateful public-key cryptosystems: How toencrypt with one 160-bit exponentiation. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 2006, pp. 380–389. ACM Press, October/November 2006
9. Bellare, M., Poettering, B., Stebila, D.: From identification to signatures, tightly: A framework and generic transforms. Cryptology ePrint Archive, report 2015/1157 (2015). http://eprint.iacr.org/2015/1157
10. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
11. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (2011)
12. Boldyreva, A., Fehr, S., O'Neill, A.: On notions of security for deterministic encryption, and efficient constructions without random oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 335–359. Springer, Heidelberg (2008)
13. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Ray, N., Kruegel. (eds.), ACM CCS 2015, pp. 668–679. ACM Press, October 2015
14. Checkoway, S., Fredrikson, Niederhagen, R., Everspaugh, M., Green, A., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H.: On the practical exploitability of dual EC in TLS implementations. In: USENIX Security, vol. 1 (2014)
15. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
16. Dodis, Y., Ganesh, C., Golovnev, A., Juels, A., Ristenpart, T.: A formal treatment of backdoored pseudorandom generators. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 101–126. Springer, Heidelberg (2015)
17. Fuller, B., O'Neill, A., Reyzin, L.: A unified approach to deterministic encryption: new constructions and a connection to computational entropy. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 582–599. Springer, Heidelberg (2012)
18. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
19. Heninger, N., Durumeric, Z., Wustrow, E.: Halderman. Mining your PS and QS: Detection of widespread weak keys in network devices. In: USENIX Security Symposium, pp. 205–220 (2012)

20. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-random generation from oneway functions (extended abstracts). In: 21st ACM STOC, pp. 12–24. ACM Press, May 1989
21. Koblitz, N., Menezes, A.: The random oracle model: a twenty-year retrospective. Cryptology ePrint Archive, report 2015/140 (2015). http://eprint.iacr.org/2015/140
22. Krawczyk, H.: LFSR-based hashing and authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
23. Vitek, J., Naccache, D., Pointcheval, D., Vaudenay, S.: Computational alternatives to random number generators. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 72–80. Springer, Heidelberg (1999)
24. Paterson, K.G., Schuldt, J.C.N., Sibborn, D.L.: Related randomness attacks for public key encryption. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 465–482. Springer, Heidelberg (2014)
25. Pornin, T.: Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA). IETF RFC 6979 (2013)
26. Raghunathan, A., Segev, G., Vadhan, S.: Deterministic public-key encryption for adaptively chosen plaintext distributions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 93–110. Springer, Heidelberg (2013)
27. Ristenpart, T., Yilek, S.: When good randomness goes bad: Virtual machine resetvulnerabilities and hedging deployed cryptography. In: NDSS 2010. The InternetSociety, February/March 2010
28. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002, pp. 98–107. ACM Press, November 2002
29. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg (2004)
30. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
31. Schnorr, C.-P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
32. Yilek, S.: Resettable public-key encryption: how to encrypt on a virtual machine. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 41–56. Springer, Heidelberg (2010)
33. Zuckerman, D.: Simulating Bpp. using a general weak random source. In: 32nd FOCS, pp. 79–89. IEEE Computer Society Press, October 1991