Skye: A Fast KDF based on Expanding PRF and its Application to Signal

Amit Singh Bhati[1], Antonín Dufka[2], Elena Andreeva[3], Arnab Roy[4], and Bart Preneel[1]

[1] imec-COSIC, KU Leuven, Belgium, {amitsingh.bhati,bart.preneel}@esat.kuleuven.be
[2] Masaryk University, Brno, Czech Republic, dufkan@mail.muni.cz
[3] Technical University of Vienna, Austria, elena.andreeva@tuwien.ac.at
[4] Alpen-Adria University, Klagenfurt, Austria, arnab.roy@aau.at

Abstract. A Key Derivation Function (KDF) generates a uniform and highly random key-stream from weakly random key material. KDFs are broadly used in various security protocols such as digital signatures and key exchange protocols. HKDF, the most deployed KDF in practice, is based on the *extract-then-expand* paradigm. It is presently used, among others, in the Signal Protocol for end-to-end encrypted messaging.

HKDF is a generic KDF for general input sources and thus is not optimized for source-specific use cases such as key derivation from Diffie-Hellman (DH) sources (i.e. DH shared secrets as key material). Furthermore, the sequential HKDF design is unnecessarily slow on some general-purpose platforms that can benefit from parallelization.

In this work, we propose a novel, efficient and secure KDF called Skye. Skye follows the *extract-then-expand* paradigm and consists of two algorithms: efficient deterministic *randomness extractor* and *expansion* functions. Instantiating our extractor for dedicated source-specific (e.g. DH sources) inputs leads to a significant efficiency gain over HKDF while maintaining the security level. We provide concrete security analysis of Skye and both its algorithms in the standard model.

We provide a software performance comparison of Skye with the AES-based expanding PRF ButterKnife and HKDF with SHA-256 (as used in Signal). Our results show that in isolation Skye performs from 4x to 47x faster than HKDF, depending on the availability of AES or SHA instruction support. We further demonstrate that with such a performance gain, when Skye is integrated within the current Signal implementation, we can achieve significant overall improvements ranging from 38% to 64% relative speedup in unidirectional messaging. Even in bidirectional messaging, that includes DH computation with dominating computational cost, Skye still contributes to 12-36% relative speedup when just 10 messages are sent and received at once.

Keywords: KDF · Deterministic Extraction · Extract-then-Expand · HKDF · X3DH · Signal · Expanding PRF · PRF-PRNG · Randomness Amplification

1 Introduction

A Cryptographic Key Derivation Function (KDF) outputs a uniform and "highly" random arbitrarily long key-stream when provided with a non-uniform or "weak" random key input. KDFs generate randomness for encryption, digital signatures, key exchange protocols, etc. HKDF [28] was introduced in 2010 by Krawczyk et al. as a secure, generic cryptographic KDF and by now is the most deployed KDF. It follows the extract-then-expand paradigm [28]. For a given non-uniform or "weak" random key, HKDF extracts a relatively small but uniform and "highly" random string via an internal *randomness extractor*. Then, the extracted value is passed to a *randomness expander*: a pseudorandom function PRF with variable length output. The output is a uniform and "highly" random key-stream. HKDF is used in the Signal Protocol [35] for end-to-end encrypted messaging where it generates the message keys. There HKDF is used in conjunction with the triple Elliptic-curve Diffie–Hellman handshake [36] (X3DH) key agreement protocol. A number of applications of end-to-end encryption also make use of Signal and internally call the X3DH handshake and a secure (H)KDF to establish a key for later cryptographic use. Examples are the popular instant messaging apps WhatsApp [34], Facebook Messenger [1], Skype [31], Allo [33], Status [5], Secure Chat, Viber and Forsta. HKDF is a main component in the Noise Protocol Framework (NPF) [39], Message Layer Security (MLS) [3], and is used in TLS1.3.

HKDF is a key derivation function for processing general input-sources of some desired min-entropy. However, in many applications the sources are predefined, fixed and contain a nice algebric structure, such as the triple Elliptic-curve Diffie–Hellman handshake [36] (X3DH) protocol in Signal. As a generic or non-source-specific KDF, HKDF meets basic security and performance goals, yet for concrete use cases, it might not be best optimized for: 1. source-specific (randomness) extraction; 2. performance on common platforms; 3. reliance on weaker assumptions than the random oracle for the SHA-256 hash function in HKDF. A dedicated KDF can efficiently leverage the structure of the input randomness source during the extraction phase to optimize the process. The possibility of constructing (input) source-specific extractor in a KDF under the extract-then-expand paradigm, was discussed in [28]. Concrete examples here are the works on deterministic randomness extraction from Diffie-Hellman schemes [15, 23].

To achieve secure and more efficient randomness expansion, a candidate building block is one that, unlike SHA-2, naturally expands its inputs. Cryptographically, such a function needs to come with some pseudorandom properties or behave as a fixed output length expanding pseudorandom function (PRF). Forkciphers [8] are such expanding functions, yet, they come with extra functionalities of inversion and reconstruction that are not necessary for randomness expansion and limit the PRF security to birthday-bound (in the block size). The recently proposed PRF ButterKnife [7] is a fixed-length expanding PRF design and a natural KDF building block candidate. ButterKnife takes 256-bit inputs and produces 1024-bit outputs. It is based on AES and Deoxys-BC [26]. Its structure allows its implementation with AES native instructions (NI) on all AES-NI supporting processors. ButterKnife is proven secure both generically and analyzed cryptanalytically in [7]. Its security is further backed by the cryptanalysis results for AES-PRF [21, 38] and Deoxys-BC [17, 26, 30, 44].

1.1 Our Results

In this work we propose Skye, a novel extract-then-expand KDF based on an expanding PRF. We provide a detailed security analysis of Skye and demonstrate empirically the efficiency advantages of Skye over HKDF, both directly and when used in Signal. We note that Skye applications are not limited to Signal and provide a discussion on possible Skye applications. Our contributions are:

Deterministic extension for randomness extraction. We build a novel, generic and deterministic function DExt_f that aggregates the amount of extracted randomness for any randomness extractor f for multiple and independent input samples. We prove that the outputs of DExt_f are indistinguishable from random binary strings (see Sec. 7). Considering the randomness aggregation and extraction of the X3DH handshake of Signal input samples of multiple (three or four) DH shared secrets, DExt_f handles multiple independent samples and is hence well-suited for optimal randomness extraction. In this work, DExt_f is developed to be used in Skye for key derivation but we emphasize that this novel result is independent and quite interesting on its own.

Secure DExt_f instantiation for DH samples. We provide a 128-bit secure simple and efficient *deterministic extractor* as an instantiation of DExt_f. This is achieved by choosing a DH source-specific extractor function to instantiate f in DExt_f. Applying the above-mentioned general result (in Sec. 7), we construct an extractor with a higher security margin from the msb/lsb (most/least significant bits based) extractor function for Diffie-Hellman (DH) schemes in [15]. We prove the security of our instantiation by combining the analysis of the generic DExt_f and the security of msb/lsb based extractor [15] (see Sec. 7.3).

Secure randomness expander. The outputs of the deterministic extractor are processed by a variable-output-length (VOL) PRF or randomness expander FExp. Two approaches to construct randomness expanders are presented in [28] – based on a counter or feedback encryption modes. HKDF uses as a randomness expander, a keyed feedback mode over the HMAC [29] pseudorandom function (PRF). To avoid the frequent rekeying of the feedback mode and the consequent speed reduction for the target 128-bit security level, we adopt a counter mode-like approach. FExp is motivated by the CTR$ [40] encryption mode and it benefits from the expanding PRF function, as opposed to a block cipher. The FExp design achieves both security and efficiency improvements over the CTR$ [40] mode in the randomness expansion context. When compared to the CTR$ mode, FExp accommodates larger and arbitrary (not necessarily random) inputs and provides full $n = 128$-bit security, as opposed to the birthday-bound $n/2 = 64$-bit, when instantiated with ButterKnife [7]. We give a security proof of FExp via the notion of indistinguishability from random binary strings, a notion equivalent to the IND$ [40] notion (see Sec. 8).

Skye: secure and more efficient alternative to HKDF in X3DH based applications. In Sec. 5, we propose the Skye scheme as a dedicated KDF following the extract-then-expand approach and using the two above-mentioned novel extractor and expander functions DExt_f and FExp, respectively. $\mathsf{Skye}[f, \mathsf{PRF}_s]$ is based on two underlying primitives - a weak source-specific extractor f (processed under DExt_f extension) and an expanding PRF PRF_s (processed under FExp mode). We consider DH samples over Curve25519 as the input source to exemplify the performance and concrete security gain of Skye over HKDF and instantiate f with lsb. In Sec. 9, we show that when Skye is instantiated with ButterKnife, it achieves 128-bit CCS security [16, 28]. HKDF [28] is also shown secure under the CCS security notion and informally, CCS-security is defined as the indistinguishability of the KDF outputs from truly random strings under a chosen input attack. Here, we prove the security of Skye under a more practical (non random oracle) assumption than HKDF.

We provide software performance comparison (in Sec. 6) of Skye with HKDF under their standard functionalities. Our results show that in isolation, Skye performs from 4x to 47x faster than HKDF in various settings depending on the availability of native instruction sets. We then consider the Signal to further illustrate the performance benefits of Skye. We integrate Skye within the current implementation of Signal [2] and show that under various settings with and without available native instruction sets, Skye is able to provide 38-64% relative speedup in unidirectional messaging. In bidirectional messaging, where cost is dominated by the DH computation, the speedup depends on the number of messages sent at once. With a single message, Skye achieves 3%-11% relative speedup; after sending just 10 messages, the speedup rises to 12-36%, and with a further increase in the number of messages, it converges to the unidirectional speedup.

1.2 Related Works

Randomness amplification. The result of Maurer et al. [37, Corollary 2] can be applied to obtain similar security as DExt_f by iterating a set of independent input samples (with some weak extractor f) and defining the $\star$ operation is the XOR. Yet, our design approach is more general by allowing us to optimally extract (up to 100%) more randomness than that existing solution. For example, consider v input samples each providing w bits of extracted randomness and λ bits of security when passed through some weak extractor f. Now, if we need a higher security of $c\lambda$ bits for some integer c, then the solution of [37] provides randomness only up to $(v/c)w$ bits. Our solution DExt_f offers up to $\lfloor (v-c)/\lceil c/2 \rceil \rfloor + 1)w$ bits of security which is optimal (see App. B) for various values of (v, c).We generalize the result of Maurer et al. [37, Corollary 2] by offering a relatively simple proof that: 1. increases confidence in the existing analysis 2. generalizes and improves the bound (optimal) for reasonable/practical sample sizes.

Related KDF notion. In the formal analysis of the Double Ratchet protocol, Alwen et al. [4] formalized a KDF syntax called PRF-PRNG and it's security notion called P. HKDF also satisfies this notion. For source-specific samples, our KDF Skye targets the stronger KDF notion of CCS-security, which implies the P-security when KDF is used as a PRF-PRNG (in the context of Signal). We refer the reader to App. E for more details on the PRF-PRNG syntax, how to convert a standard syntax KDF into a PRF-PRNG and a formal claim with full proof of CCS to P security reduction.

Alwen et al. [4] provide a standard model proof for the Double Ratchet and Signal Protocol, however, their analysis is not RO-free when the KDF is instantiated with HKDF or alternatives with idealized assumptions (see App. E). This leads us to naturally consider the possibility of designing a KDF for applications like (but not limited to) the Signal Protocol that is RO-free and is equally or more efficient than the current HKDF solution. The Skye KDF follows the standard KDF syntax [28] and is proven to be CCS [16, 28]-secure in the standard model.

2 Notations

Strings. All strings are binary strings. The set of all strings of length n (for a positive integer n) is denoted by $\{0,1\}^n$ and the set of all strings of all possible lengths is denoted by $\{0,1\}^*$. We denote by $\mathrm{Func}(m,n)$ the set of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$. For a string X of ℓ bits, we let $X[i]$ denote the i^{th} bit of X for $i=0,\ldots,\ell-1$ (starting from the left) and $X[i\ldots j] = X[i]\|X[i+1]\|\ldots\|X[j]$ for $0 \le i < j < \ell$. We let $\mathsf{msb}_\ell(X) = X[0\ldots(\ell-1)]$ denote the ℓ leftmost (most significant) bits of X and $\mathsf{lsb}_r(X) = X[(|X|-r)\ldots(|X|-1)]$ the r rightmost (least significant) bits of X, such that $X = \mathsf{msb}_\chi(X)\|\mathsf{lsb}_{|X|-\chi}(X)$ for any $0 \le \chi \le |X|$. Given an xn-bit string X, we let $X_1,\ldots,X_x \xleftarrow{n} X$ denote a partitioning of X into n-bit blocks, such that $|X_i| = n$ for $i = 1,\ldots,x$.

Miscellaneous. The symbol $\perp$ denotes an error signal, or an undefined value. We denote by $X \xleftarrow{\$} \mathcal{X}$, a sampling of an element X from a finite set $\mathcal{X}$ following the uniform distribution. We use lexicographic comparison of tuples of integers; i.e. $(i',j') < (i,j)$ iff $i' < i$ or $i' = i$ and $j' < j$. We denote the set of natural numbers by $\mathbb{N}$. For a matrix M, we use $M[ij]$ to denote the ij^{th} entry of M. For equations E_is with $1 \le i \le a$ (for some $a \in \mathbb{N}$), we use the indexed set $S = \{E_i | 1 \le i \le a\}$ to denote the system of equations E_is.

3 Key Derivation Function

KDFs derive one or more *cryptographically secure* secret keys (of any fixed length) from a source of initial key material (IKM) that can contain some good amount of randomness but is not distributed uniformly. The notion of cryptographically secure keys is usually associated with pseudorandom keys, i.e. keys that are computationally indistinguishable from a uniformly random string of the same length. Formally, a source of IKM can be defined as follows.

Definition 1 (Source of IKM [28]). *A source of initial keying material (or simply source) Σ is a two-valued probability distribution $(Z, \mathcal{C}_Z)$ generated by an efficient probabilistic algorithm (we will refer to both the probability distribution as well as the generating algorithm by Σ).*

Here, Z values are used to denote the random secret IKM samples with required entropy (e.g., DH shared secrets) whereas $\mathcal{C}_Z$ provides the public info associated to Z (e.g., DH public keys and group parameters). See Sec. 9 for more details.

A KDF function tackles the case when the initial key material is not pseudorandom or uniformly random, e.g. the initial key material is obtained from a weak process that uses renewable sources of randomness, a weak random number generator, random sampling over a group or DH values computed in a (key-exchange) protocol. In these settings a KDF function is constructed using the so-called *extract-then-expand* paradigm.

3.1 Extract-then-Expand Paradigm

A KDF following the extract-then-expand paradigm has two components: 1. a randomness extractor Ext that extracts a fixed-length pseudorandom key K from an "imperfect" source of initial key material; 2. randomness expander Exp that expands K to a variable-length output. The latter is usually built using a regular pseudorandom function (PRF) with output extension via counter or feedback encryption modes [28,40]. Ext produces "close-to-random" outputs (in a computational or statistical sense) from an input that is sampled from the corresponding source key distribution. The extraction may have an additional non-secret input or salt value, that is either randomized or constant. When it is constant, the extractor and the corresponding KDF are called *deterministic extractor* and *deterministic KDF*, respectively. The expander Exp uses K to produce cryptographic keys K_{kdf} of a specified length. Exp takes the output length parameter as one of the inputs.

A KDF scheme is defined over the inputs: a source key Z, the extractor salt salt (null or constant), the length ℓ of key bits to be output by KDF, and a context variable or auxiliary info string γ (may be null). The latter string includes key-related information that is uniquely (and cryptographically) bound to the produced output. For example, includes information about the application or protocol calling the KDF, session-specific information like nonces, time, session identifiers, etc. The KDF evaluation is defined as: $K = \mathsf{Ext}(\mathsf{salt}; Z), K_{kdf} = \mathsf{Exp}(K; \gamma; \ell)$.

HKDF key derivation function.

In the expansion phase the output is $K_{kdf} = K_1 \| K_2 \| \ldots \| K_t[1 \ldots \ell]$ where $t = \lceil \ell / n_{\mathrm{HMAC}} \rceil \in \mathbb{N}$, ℓ is the desired output length and n_{HMAC} is the output size. K_i are obtained as:

$$K_1 = \mathrm{HMAC}(K, \gamma \| 0), K_{i+1} = \mathrm{HMAC}(K, K_i \| \gamma \| i)\,, \quad 1 \leq i < t,$$

where γ denotes the context variable.

3.2 Deterministic Multi-sample KDF

A KDF is *multi-sample* if it takes multiple independent samples from its input source and combines those for relatively large and highly random output with the same or increased security than its single-sample variant. Formally, a deterministic multi-sample KDF $\Pi : (\{0,1\}^*)^v \times \Gamma \times \mathbb{N} \rightarrow \{0,1\}^*$ takes three arguments; a set Z of v values Z_is (binary strings) sampled from a source of keying material (defined in Def. 1), a context variable or auxiliary info γ (optional, i.e., null string or a constant) and the desired output length ℓ. The function returns an ℓ-bit binary string K_{kdf}. HKDF [28] is a deterministic multi-sample KDF scheme when its salt value is set to a constant.

4 KDF calls in the Signal Protocol

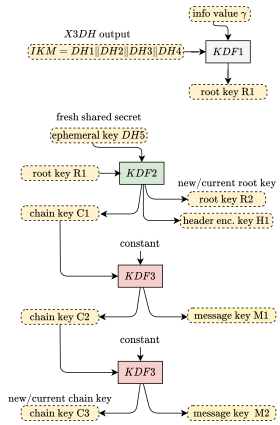
The security in the Signal Protocol is due to the use of the Double Ratchet algorithm [35], the X3DH (triple Elliptic-curve Diffie–Hellman) handshake [36], the key derivation function HKDF [28], and an AEAD mode. Signal Protocol is typically implemented with Curve25519 [11], AES-256, and HMAC-SHA256 as its underlying cryptographic primitives.

Signal as defined in [35]. It makes KDF calls for three different purposes (see Fig. 1a): 1. to generate a **root key** from the X3DH outputs and the info value γ; 2. to generate a **chain key**, a **header encryption key** and a **new root key** from the present root key and an **ephemeral DH shared secret key**; and 3. to generate a **message key** and a **new chain key** from the old chain key and a predefined constant. The three processes are realized by KDF1, KDF2 and KDF3, respectively, which differ in the types and sizes of inputs and outputs. KDF1 is used first and only once in a session (between two users) to generate the initial root key. When a user sends $s > 0$ many concurrent messages to another user, the root key is used along with a fresh DH shared secret as an input to KDF2 to produce a chain key and update the root key. Then, s many iterative KDF3 calls are made with the i^{th} chain key as input to generate the i^{th} message keys and the $i+1^{th}$ chain key for $1 \leq i \leq s$. Fig. 1a illustrates the sequence of calls with its input and output values.

Current implementation libraries [2] of Signal use HKDF [28] for each of these KDF calls. When instantiated with HKDF, the type and size of each input of Signal's KDF calls in terms of HKDF's arguments is provided in Table 1b (for 128-bit secure version).

Skye as a more efficient alternative to HKDF for Signal.

I. For all KDF1 calls in Signal, the salts for HKDF are set to a constant and the underlying SHA-256 is modeled as a random oracle (RO) to achieve the claimed



(a)

KDF type	Input type	Corr. argument in HKDF	Size (in bits)
KDF1	Info value γ	context variable	256
	X3DH output	IKM	256 $\times$ (3 or 4)
	Null string	salt	256
KDF2	Null string	context variable	256
	Ephemeral DH key	IKM	256
	root key	salt	256
KDF3	Null string	context variable	256
	constant	IKM	256
	chain key	salt	256

(b)

Fig. 1: (a) The three types of KDF calls in the Signal Protocol. Here IKM represents the initial key material which is defined as the concatenation of (3 or) 4 (depending upon the availability of the corresponding one-time prekeys for $DH4$, see [35]) DH shared secrets from the X3DH handshake. (b) Types and sizes of Signal's KDF's inputs in terms of HKDF-HMAC-SHA-256's arguments.

security level (see Lemma 10 and its following paragraphs in [28]). While this is a requirement for general HKDF input sources, it is possible to relax the RO requirement. Further, the reliance on specific randomness sources aids towards a simplified and efficient extraction KDF phase in Signal. In the expansion phase on the other hand, one can benefit both security- and efficiency-wise from an in-built expanding cryptographic primitive with pseudorandom properties, such as the ones captured by the forkcipher [8] (e.g., ForkSkinny [8]; based on the lightweight ISO standard SKINNY [9]), multiforkcipher [6] and expanding PRF [7] (e.g., Butterknife [7]; based on AES and Deoxys-BC [26]) notions. Note that HKDF currently uses a "compressing" primitive instead (HMAC-SHA-256) in a key feedback mode for the expansion phase.

II. Fig. 1a shows that only KDF1 requires the use of both the randomness extraction and expansion. KDF2 and KDF3 are made over pseudorandom keys that are generated by the prior KDF calls. Specifically, the root keys that are fed to KDF2 calls are pseudorandom keys as they are generated by the KDF1 calls and the chain keys that are fed to KDF3 calls are pseudorandom keys as they are generated by the KDF2 calls. In such cases, the extraction phase becomes redundant and the HKDF for KDF2 and KDF3 can be replaced by calling a fast and secure PRF (using the pseudorandom keys) that gives the required expansion.

Addressing these points, we propose Skye. We replace HKDF with Skye as a KDF for KDF1 and as its expanding component FExp for the KDF2 and KDF3 calls. As a consequence, we improve the performance of Signal at full 128-bit security level. For simplicity, we limit our discussion to the 128-bit secure variant. Our results can be analogously be lifted to 224-bit security level under suitable instantiation.

5 Skye: An Expanding PRF based KDF

In this section, we define Skye. Skye uses an expanding pseudorandom function PRF_s. $\mathsf{PRF}_s : \{0,1\}^k \times \{0,1\}^{2n} \to \{0,1\}^{sn}$ is a symmetric primitive that transforms a fixed length ($2n$-bit) input X into a larger fixed-length (sn-bit with $s \geq 2$) output Y via a secret key $K \in \mathcal{K}$ of k bits.

Definition 2 (PRF Security). *For* $\mathsf{PRF}_s : \{0,1\}^k \times \{0,1\}^{2n} \to \{0,1\}^{sn}$*, let* $\mathcal{A}$ *be an adversary whose goal is to distinguish* $\mathsf{PRF}_s(K,\cdot)$ *and a uniform random function* $R(\cdot) : \{0,1\}^{2n} \to \{0,1\}^{sn}$ *by their oracle access. The* prf*-security of* PRF_s *is defined as:*

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF}_s}(\mathcal{A}) = \big|\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{PRF}_s(K,\cdot)} \Rightarrow 1] - \Pr[R \xleftarrow{\$} \mathrm{Func}(2n, sn) : \mathcal{A}^{R(\cdot)} \Rightarrow 1]\big|.$$

Description of Skye. Skye is a deterministic multi-sample KDF that follows the extract-then-expand approach by using a deterministic source-specific extractor DExt_f and a randomness expander FExp, respectively. In Signal we instantiate DExt_f with $\mathsf{DExt}_{\mathsf{lsb}}$. Also, for simplicity, in the remainder of the paper, consider Skye with $f = \mathsf{lsb}$. We note that for any other source (i.e. applications beyond multi-DH sources) with a suitable weak extractor f_0, Skye can be simply defined by reinstantiating f with f_0 instead of lsb. For the general design and security analysis of DExt_f, we refer the reader to Sec. 7.

Skye in Signal. The inputs to Skye in Signal are:

- the set Z of $v = 3$ or 4 (depending on the available one time prekeys, see [35]) independent samples DH_i ($i = 1, \ldots, v$) of Diffie-Hellman (DH) shared secrets from the source group G defined over Curve25519 where each sample is of length $2n$;
- the auxiliary info $\gamma \in \{0,1\}^{2n}$;
- $\ell \in \mathbb{N}$ denoting the desired output length;

Skye outputs $K_{exp} \in \{0,1\}^{\ell}$ as shown in Fig. 2. $\mathsf{DExt}_{\mathsf{lsb}}$ takes the inputs Z and an integer k denoting the output size of $\mathsf{DExt}_{\mathsf{lsb}}$ and produces a k-bit output K_{ext}. Then, K_{ext} together with γ and ℓ are input to the FExp. FExp uses a fixed PRF_s to produce the ℓ-bit K_{exp}. $\mathsf{DExt}_{\mathsf{lsb}}$ and FExp are given in Fig. 3a and 3b, respectively with pseudocode of FExp provided in Fig. 4.

Although in Fig. 2 k appears as an external input to $\mathsf{DExt}_{\mathsf{lsb}}$, for a fixed PRF_s it is tied to its keysize. Hence, k is not a part of the global inputs to $\mathsf{Skye}[f, \mathsf{PRF}_s]$. For simplicity, we also drop f from the global inputs whenever it is fixed to lsb, i.e. when working with DH-sources.

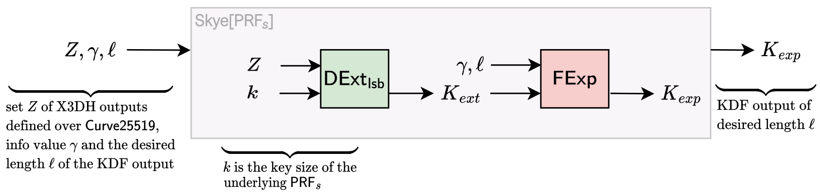
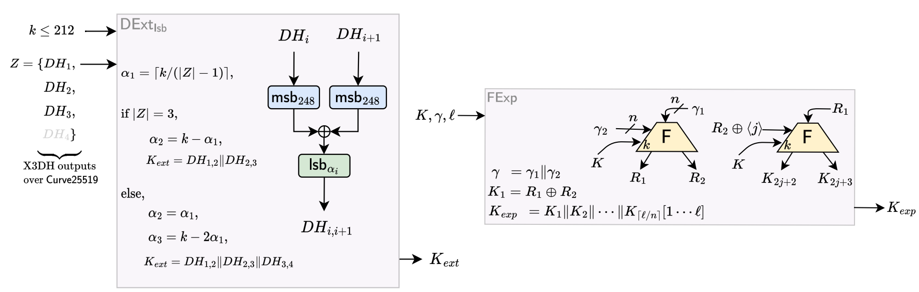


Fig. 2: Skye KDF in the context of Signal. See Fig. 3a and 3b for the internals of functions DExt$_{\mathsf{lsb}}$ and FExp.

Parameters	Argument types	Sizes
set Z	DExt$_{\mathsf{lsb}}$ input	$256 \times (3 \text{ or } 4)$ bits
K_{ext}	DExt$_{\mathsf{lsb}}$ output	$k = 128$ bits
info value γ	FExp input	256 bits
K_{exp}	FExp output	ℓ bits

Table 1: Parameter sizes of input-output arguments in DExt$_{\mathsf{lsb}}$ and FExp components of Skye[lsb, ButterKnife].



(a) (b)

Fig. 3: (a) DExt$_{\mathsf{lsb}}$: A deterministic extractor that can extract up to 212 bits of randomness with 128 bits of security from 3 or 4 random, independent and fresh DH shared secrets that are X3DH outputs over Curve25519. Here msb and lsb are the most and least significant bits functions, respectively, as defined in Sec. 2. (b) FExp mode of randomness expansion. Here K denotes the secret key to the PRF$_s$ and $\langle j \rangle$ is a suitable n-bit binary encoding of j (e.g., $0^{n-\lceil \log_2(j) \rceil} \| j$). Here each concatenation separates n-bit (block size) strings.

We instantiate PRF$_s$ in Skye with ButterKnife with $n = 128$ bits and $s = 8$. ButterKnife is well-suited due to its AES internal structure and as a result benefiting from the AES native instructions (NI) on all supporting processors. Additionally, it has a very large expansion of $s = 8$ for performance gains and efficient key-scheduling [7] that saves the cost of frequent key scheduling. ButterKnife comes with dedicated cryptanalysis and a generic proof of security [7] and is backed by further cryptanalytic results for AES-PRF [21, 38] and Deoxys-BC [17, 26, 30, 44].

$\mathsf{FExp}(\mathrm{K},\gamma,\ell)$

$//K \in \mathcal{K}, \gamma \in \{0,1\}^{2n}, \ell \in \mathbb{N}$

$K_1, K_2, \ldots, K_s \leftarrow \mathsf{PRF}_s(K,\gamma), j_{\mathsf{last}} \leftarrow \lfloor \lceil \ell/n \rceil / s \rfloor, s_{\mathsf{last}} \leftarrow \lceil \ell/n \rceil - j_{\mathsf{last}} s$
for j **in** $[0 \ldots j_{\mathsf{last}} - 2]$ **do**
$\quad K_{(j+1)s+1}, K_{(j+1)s+2}, \ldots, K_{(j+2)s} \leftarrow \mathsf{PRF}_s(K, K_1 \| (K_2 \oplus \langle j \rangle))$
$K_{j_{\mathsf{last}}s+1}, K_{j_{\mathsf{last}}s+2}, \ldots, K_{\lceil \ell/n \rceil} \leftarrow \mathsf{PRF}_{s_{\mathsf{last}}}(K, K_1 \| (K_2 \oplus \langle j_{last} \rangle))$
$K_{exp} \leftarrow K_1 \| \ldots \| K_{\lceil \ell/n \rceil}[1 \ldots \ell]$
return K_{exp}

Fig. 4: (b) FExp mode pseudocode.

The parameter sizes of $\mathsf{DExt}_{\mathsf{lsb}}$ and FExp with ButterKnife are described in Table 1.

6 Software Performance of Skye

In this section, we present the performance evaluation of our implementation of Skye in comparison to HKDF both in an isolated setting and within the current Signal Protocol [2]. We use the Rust implementation of HKDF from the current Signal implementation. Skye's implementation is also done in Rust, and it is instantiated with ButterKnife. Our measurements are on the x86_64 platform and we consider performances both with and without using relevant instruction set extensions (AES-NI and SHA-NI). All measurements were performed with AMD Ryzen 7 5800X CPU.

6.1 Isolated Performance

To provide a direct comparison, we measured the execution time[5] of sending up to n messages in a sequence in Signal. This benchmark includes the computation of KDF1, KDF2, and then n executions of KDF3, but not the time to compute the inputs to the KDF (i.e., they are constants), nor the time to perform any follow-up operations like message encryption with the derived key. On average, our implementation with enabled AES-NI and SHA-NI extensions achieves at least 91% (or ≥11x) speedup, while without the extensions at least 76% (or ≥4x) speedup. On platforms that support AES-NI but not SHA-NI[6], the speedup is at least 98% (or ≥47x) on average, as HKDF cannot utilize AES-NI, while Skye instantiated with ButterKnife , it can. Full results are shown in Table 2.

[5] We measure wall-clock time with ns precision to be consistent with benchmarks present in the Signal implementation.

[6] SHA-NI support was released for public markets in 2017-18 with Intel's Goldmont microarchitecture. All processors and devices before that and many after that do not have the support for SHA-NI.

n	Skye with NI	HKDF with NI	Skye without NI	HKDF without NI
1	157	1739	1794	7404
2	218	2595	2666	11014
3	273	3459	3542	14640
4	336	4318	4414	18275
5	389	5179	5289	21901
6	453	6035	6164	25535
7	505	6894	7037	29160
8	570	7753	7919	32785
9	622	8617	8787	36406
10	688	9470	9667	40036

Table 2: The mean time (in ns) required to generate n message keys using Skye and HKDF with and without support for AES-NI and SHA-NI extensions. The measurements were repeated 10^4 times.

6.2 Performance within Signal

We integrated Skye within the current Signal implementation [2]. First, we present the results from the extended benchmarks of the original Signal code. In these settings, we assume that session initialization has already been performed (this includes the one time cost of X3DH computation and KDF1 call), and we measure the subsequent communication.

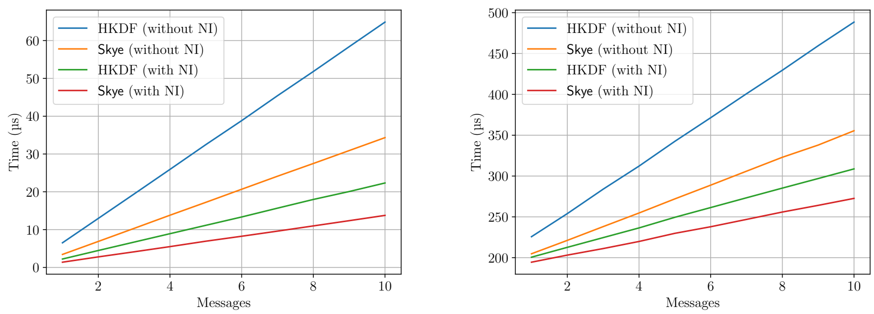
The Signal library also supports group messaging however, each group message is handled as a direct message to each receiver in the group, i.e., if there are N members in the group, Signal client sends N messages individually encrypted with the derived ratchet key of each participant. Hence, we focus on two-party messaging for analyzing performance in both direct and group messaging.

Unidirectional messaging: Here one party sends messages to another in a sequence without any reply in between. Fig. 5a presents the time it takes for the party to encrypt n messages.[7] The relative speedup is independent of n and on average equal to 38% with NI instructions, 47% without NI instructions, and 64% with AES-NI but not SHA-NI[8]. For complete results, see Table 4 in App. F.

Bidirectional messaging: The previous setting does not include the cost of DH-ratchet, hence we consider: a party sends n messages (in sequence) to another party, which then replies with n messages. In Fig. 5b, we present the time required to encrypt and decrypt the sent and received messages, respectively. Since the DH computation cost diminishes the impact of the rest of the computation (especially when the KDF calls are a few), we gain only around 3% with single message i.e. $n = 1$ with NI instructions, but (as expected) with more

[7] We use one block test messages of size 128 bits (i.e. 16 characters).

[8] Note that in this case AES-NI and SHA-NI were also used for message encryption and authentication so this number (64%) cannot be computed from the figure and is based on the full measurement.



(a) (b)

Fig. 5: (a) The mean time (in µs) required to encrypt n messages in the Signal Protocol implementation using Skye and HKDF. (b) The mean time (in µs) required to encrypt (on sender side) then decrypt (on receiver side) n messages, and then encrypt (on receiver side) and decrypt (on sender side) another n messages in the Signal Protocol implementation using Skye and HKDF. In both cases, the measurements were performed for variants with and without AES-NI and SHA-NI support. The encrypted messages were of less than 16 characters. All measurements were repeated 10^3 times.

messages, the speedup improves and comes closer to the unidirectional case. For example, the speedup quickly improves here to 12% for $n = 10$. Similarly, without NI, we gain a speedup of around 9% with $n = 1$, and with $n = 10$, up to 27%. Lastly, with the partial extension support (with AES-NI but not SHA-NI) we get 11% with $n = 1$ that goes to 36% with $n = 10$. For complete results, see Table 5 in App. F.

Speedup due to the expanding PRF. In the unidirectional experiment, 81%-93% of the total performance gain is due to the use of an expanding PRF (ButterKnife) in place of a compression function and the rest 19%-7% is due to the replacement of KDF2 and KDF3 calls by FExp. In the bidirectional experiment, with 10 messages, 78%-86% of the total performance gain is due to the use of an expanding PRF and the rest 22%-14% is due to the KDF2 and KDF3 replacements. These results illustrate the explicit performance advantage of adding an expanding PRF.

7 Randomness Extraction Phase

We first recall some preliminary definitions that are required for presenting our results.

Definition 3 (Statistical Distance). *Let X and Y be two random variables taking values from a finite set $\mathcal{X}$. The statistical distance between X and Y is* *the value of the following expression:*

$$\mathsf{SD}(X,Y) = \frac{1}{2}\sum_{x\in\mathcal{X}} |\Pr[X=x] - \Pr[Y=x]|\,.$$

7.1 Elliptic Curve Decisional Diffie-Hellman (ECDDH) Problem [12]

Let G be a public cyclic subgroup (of size a prime q) of an elliptic curve group $\mathcal{E}(\mathbb{F}_p)$ over the finite field $\mathbb{F}_p$ of size prime p. Let P be a randomly chosen generator of G, then for some secret and randomly chosen $a, c \in \mathbb{Z}_q^*$ and a randomly chosen $Q \in G$, it is hard to determine whether $Q = acP$ or $Q \neq acP$ when provided with P, aP, cP and Q. Formally, we define the advantage of an adversary $\mathcal{A}$ against solving ECDDH in G as

$$\begin{aligned}\mathbf{Adv}_G^{\mathsf{ecddh}}(\mathcal{A}) = |\Pr[&\mathcal{A}(P,aP,cP,acP)=1] \\ &- \Pr[\mathcal{A}(P,aP,cP,Q)=1]|\end{aligned}$$

where the probability is over the random choices of $a, c \in \mathbb{Z}_q^*,\ P, Q \in \mathbb{G}$ and the internal randomness of $\mathcal{A}$.
The ECDDH assumption (with respect to G) states that for all *efficient* adversaries, the value of advantage $\mathbf{Adv}_G^{\mathsf{ecddh}}(\mathcal{A})$ is reasonably *small*.

In the remaining section, we now show how to build a strong extractor from a weak extractor and provide its security analysis. Formally, an extractor is defined as follows:

Definition 4 ((X,ϵ)-extractor [28]). *Let X be a random variable that takes values in $\{0,1\}^x$ and U_w denote a random variable uniformly distributed in $\{0,1\}^w$ for some positive integers x and w, where U_w and X are independent. We say that for some non-negative integer d, the function $f : \{0,1\}^x \times \{0,1\}^d \to \{0,1\}^w$ is an (X,ϵ)-extractor if*

$$\mathsf{SD}((f(X,\mathsf{salt}),\mathsf{salt}),(U_w,\mathsf{salt})) \leq \epsilon$$

where the salt *is a d-bit value which is made public upon sampling.*

We note that the salt value can be a null string or constant (in case of deterministic extractors). Further, since the salt sampling is independent to f, X and U_w, for simplicity, we slightly abuse the notation and omit both salt terms from the second arguments in the expression of Def. 4 . In other words, the expression becomes

$$\mathsf{SD}(f(X,\mathsf{salt}),U_w) \leq \epsilon\,.$$

7.2 DExt: A Generic and Deterministic Extension Towards Improved Security of any Extractor

In this section, we describe how to build a stronger extractor DExt_f which can more securely extract the same or larger amount of randomness from a relatively weak extractor f given multiple independent samples. Our construction uses only simple and cheap operators like concatenation and XOR to avoid additional computational cost. Unlike the prior works based on extraction from multiple sources [18,19,22,27,42], our construction requires independence only within the samples where the source of these samples can still remain the same. An example here is multiple independent DH handshakes defined over the same source group.

Definition of DExt_f. We provide a definition of the extended extractor DExt_f in Fig. 6 as a function that takes as inputs:

- a set $Z = \{Z_1, Z_2, \ldots, Z_v\}$ of v many z-bit (when represented in binary) independent samples chosen from some public finite sets $S_1, S_2, \ldots, S_v$, respectively (all of these sets could be same) for some positive integer z;
- the desired output length k;
- a positive integer e (security parameter);
- a pair (f, ϵ) with $f : \{0,1\}^z \times \{0,1\}^d \to \{0,1\}^w$ being a (U_{Z_i}, ϵ)-extractor for some $\epsilon > 0$, positive integers d and w and all $1 \leq i \leq v$;
- a d-bit salt value salt (if any).

DExt_f then uses the (U_{Z_i}, ϵ)-extractor (for all $1 \leq i \leq v$) f and the function invXOR (defined in Def. 5) and outputs a k-bit string K_{ext} if $k \leq wb$ or $\perp$ otherwise. Here, U_{Z_i} denotes a random variable distributed according to the sampling of Z_is from the set S_i for all is, respectively, $b = \left\lfloor \frac{v-c}{\lceil c/2 \rceil} \right\rfloor + 1$ and $c = \min_i \{c_i \in \mathbb{N} |\ b_i(2\epsilon)^{c_i} \leq 2^{-e+1},\ b_i = \left\lfloor \frac{v-c_i}{\lceil c_i/2 \rceil} \right\rfloor + 1\}$. We note that the parameters c, e, b and the function f are important to define DExt_f, its design optimality (Theorem 6) and security (Theorem 1 and its following corollary).

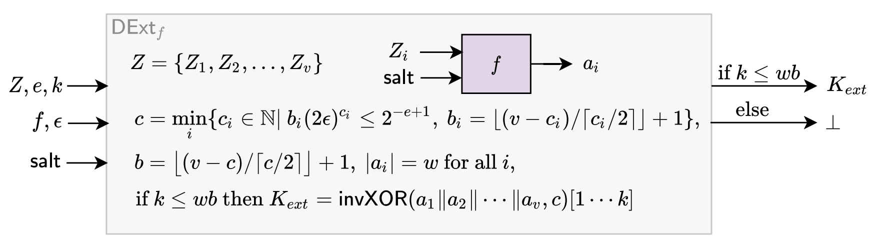


Fig. 6: DExt_f: A $(U_Z, b(2\epsilon)^c/2)$-extractor (with upto wb bits of extraction) parameterized by a relatively small and weak (U_{Z_i}, ϵ)-extractor f (with upto w bits of extraction). Here U_{Z_i} and U_Z denote random variables distributed according to the sampling of Z_is from S_is for all is and of corresponding Z from the Cartesian product of all S_is, respectively.

Definition 5 (XOR-based Involvement). *We define* invXOR *as a function that maps a* wv*-bit binary string* $a_1 \| a_2 \| \cdots \| a_v$ *where all* a_i*s are* w*-bit binary strings (equivalently elements of* $\mathbb{F}_{2^w}$ *for some integer* $w \geq 0$*) and an integer* $c < v$ *to an* wb*-bit binary string as follows:*

$$\mathsf{invXOR}(a_1\|a_2\|\cdots\|a_v, c) = \oplus_{i=1}^{c} a_i \| \oplus_{i=1+\lceil c/2\rceil}^{i=c+\lceil c/2\rceil} a_i \| \cdots \| \oplus_{i=1+(b-1)\lceil c/2\rceil}^{i=c+(b-1)\lceil c/2\rceil} a_i$$

where $b = \lfloor (v-c)/\lceil c/2\rceil \rfloor + 1$ *.*

We refer the reader to App. B for a detailed analysis and discussion on the (sub-)optimality of the DExt_f construction. This analysis shows that although finding an optimal extension for randomness extraction over arbitrary values of v, c is hard, it is possible to construct one for $v < 6$ (the relevant case in practice) and DExt_f is one example of such extensions.
With all necessary definitions in place, we are now ready to give the formal security statement of DExt_f in Theorem 1.

Theorem 1. *Let $Z_1, Z_2, \ldots, Z_v$ be v independent z-bit binary strings that are chosen from the public finite sets $S_1, S_2, \ldots, S_v$, respectively for some positive integer z. Let f be a public function with a range in $\{0,1\}^w$ for some positive integer w. Let k and c be two positive integers, such that $c \le v$ and $k \le wb$ where $b = \lfloor (v-c)/\lceil c/2\rceil \rfloor + 1$. If there exists an $\epsilon > 0$ such that f is a (U_{Z_i}, ϵ)-extractor for all i then, we have*

$$\mathsf{SD}(U_D, U_k) \le \frac{b}{2}(2\epsilon)^c .$$

Here U_{Z_i} is a random variable distributed according to the sampling of Z_is in the set S_i. U_k is a random variable uniformly distributed over $\{0,1\}^k$. U_D is a random variable distributed according to the outputs of DExt_f (of size k bits) when provided with $Z_1, Z_2, \ldots, Z_v, k, c$ and salt (if any) as inputs.

Increasing the value of c for a fixed distribution of Z_is will increase the security margin of the final outputs linearly, but will also hyperbolically decrease the maximum possible length of the final outputs. This trade-off is important and explains why it is good to leave c a free variable in the theorem (this can later be defined according to the requirements of an application). We defer the proof of Theorem 1 to App. C.1.

7.3 DExt_f Instantiation based on msb/lsb Function

In this section, we show that for applications like the Signal Protocol where the KDF source of inputs is a subgroup of an elliptic curve group and the input distribution is computationally indistinguishable from the uniform distribution over the source, one can instantiate the underlying function f of DExt_f by the functions msb or lsb. More concretely, we recall one of the two main theorems from the work of Chevalier et al. [15] on the security of lsb_k function as a deterministic extractor (this function does not require any additional salt during its evaluation) and combine it with Theorem 1 to amplify the extracted output's size and security.

Theorem 2 ($\mathsf{msb}/\mathsf{lsb}$ extraction, Theorem 14, [15]). *Let p be an ℓ_p-bit prime, G a subgroup of $\mathcal{E}(\mathbb{F}_p)$ of cardinality q generated by P_0, q being an ℓ_q-bit prime, U_G and U_k be two random variables uniformly distributed in G and $\{0,1\}^k$, respectively for some positive integer k. Then we have*

$$\mathsf{SD}(\mathsf{lsb}_k(U_G), U_k) \leq 2^{(k+\ell_p+\log_2 \ell_p)/2+3-\ell_q} .$$

By combining the results from Theorem 1 and 2 we obtain the following corollary:

Corollary 1. *Let p be an ℓ_p-bit prime, G a subgroup of $\mathcal{E}(\mathbb{F}_p)$ of cardinality q generated by P_0, q being an ℓ_q-bit prime, U_D a random variable distributed according to the outputs of $\mathsf{DExt}_{\mathsf{lsb}}$ for v many chosen uniform and independent samples from G with two positive integers k and $c(< v)$ and $b = \lfloor (v-c)/\lceil c/2 \rceil \rfloor + 1$. Let U_k be a random variable uniformly distributed in $\{0,1\}^k$. We have $\mathsf{SD}(U_D, U_k) \leq 2^{-e}$ for some positive integer e if*

$$2\ell_q - \ell_p - \log_2 \ell_p - 6 \geq \frac{2(e + \log_2 b - 1)}{c} + \left\lceil \frac{k}{b} \right\rceil .$$

In the concrete settings of the Signal Protocol, we have the source group G defined as the cyclic subgroup of Curve25519 [11] using the base point $x = 9$ (one of the NIST standards for ECC [14] targeting 128-bit security) with $\ell_p = 256$ bits and $253 \geq \ell_q \geq 252$. Further, under the ECDDH [12] assumption (see Sec. 7.1), the X3DH [36] handshake over G provides at least 3 uniform and independent group elements as DH shared secrets. Hence, from Corollary 1, we have that DExt can output upto $k = 212$ bits of randomness with security of $e = 128$ bits when provided with fresh, independent and random X3DH outputs with the least significant byte (a.k.a. the clamped [32] byte that contains three fixed bits as 0s) dropped.

Note that with the results from [15] (Theorem 2) one can achieve the same amount of randomness, i.e. $k \approx 212$ bits by concatenating the outputs of $f = \mathsf{lsb}$, but with at most security of $e = 82$ bits when provided with at least $v = 3$ uniform and independent samples from the group G.

Let us consider the following example to clarify a different implication of this result. For a fixed sample size $v = 10$, source settings $p = 256$ and $q \geq 252$, and a security parameter $e = 80$, the input size that can be extracted in the final concatenated output (computed as concatenation of the outputs of $f = \mathsf{lsb}$) is only $\approx 26\%$ (according to the results of [15]). On the other hand, the same is improved to 52% (according to our results) when evaluated with $\mathsf{DExt}_{\mathsf{lsb}}$ under the same settings.

To summarize, for a given X3DH output with 3 (resp. 4) random, independent and fresh DH samples (defined after dropping the fixed/clamped byte) as $DH1 \| DH2 \| DH3$ (resp. $DH1 \| DH2 \| DH3 \| DH4$) over $\mathsf{Curve25519}$, we have shown that under the ECDDH assumption the string $\mathsf{lsb}_{\lceil k/2 \rceil}(DH1 \oplus DH2) \| \mathsf{lsb}_{k-\lceil k/2 \rceil}(DH2 \oplus DH3)$ (resp. $\mathsf{lsb}_{\lceil k/3 \rceil}(DH1 \oplus DH2) \| \mathsf{lsb}_{\lceil k/3 \rceil}(DH2 \oplus DH3) \| \mathsf{lsb}_{k-2\lceil k/3 \rceil}(DH3 \oplus DH4)$) is indistinguishable from a k-bit uniform random string with a security of at least 128 bits when $k \leq 212$. A simplified version of DExt_f in the Signal Protocol with f defined by the lsb function is provided in Fig. 3a.

8 Randomness Expansion Phase

Randomness expansion or key-stream generation is a process to generate a large amount of random bits from relatively smaller random and secret keys. Formally, a randomness expander scheme $\Pi : \mathcal{K} \times \Gamma \times \mathbb{N} \rightarrow \{0,1\}^*$ takes a k-bit key $K \in \mathcal{K}$, the desired output length $\ell \in \mathbb{N}$ and an additional arbitrary but fixed length binary string $\gamma \in \Gamma$ as inputs and returns an ℓ-bit binary string K_{exp} as an output.

In [28], two approaches are discussed to construct randomness expanders: counter and feedback encryption mode style. HKDF uses a randomness expander that is defined using a key feedback mode over HMAC. HMAC in key feedback mode performs slow due to the frequent rekeying in HMAC calls (consequence of using feedback mode) and double hashing per HMAC call (consequence of using HMAC). In order to avoid these speed breakers, while maintaining 128-bit security level, we turn to the counter mode-like approach and consider some built-in expanding primitives such as a PRF_s with $s \geq 2$ as the underlying primitive. The known counter mode CTR$ (CTR with random IV) [40] is a good randomness expander, however, in its original form it accommodates γs only if they are chosen uniformly at random and are of size n bits (where n is the underlying block size).

Our proposal FExp is a highly efficient randomness expander that can accommodate arbitrary γs of size $2n$ bits. Further, unlike CTR$ mode which can only provide $n/2$ bits of security under the indistinguishability from random strings IND$ [40] notion, we show that FExp mode (for a given secret random key) provides full n-bit security under the same security notion. We adjust the notion under the syntax of a randomness expander scheme and denote it by gexp below.

gexp **Security.** The security of a randomness expander or in short, Exp scheme Π is defined with the help of the games **gexp-real** and **gexp-ideal** in Fig. 7. The security of Π is measured as the indistinguishability of its outputs from random strings in a chosen input attack. More precisely, given Π and an adversary $\mathcal{A}$ who interacts with either **gexp-real** or **gexp-ideal**, we define $\mathcal{A}$'s advantage at breaking the gexp security of Π as:

$$\mathbf{Adv}_{\Pi}^{\mathsf{gexp}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\mathbf{gexp\text{-}real}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{gexp\text{-}ideal}} \Rightarrow 1]\right| .$$

8.1 FExp: A PRF_s-based Randomness Expander

We provide a definition for our randomness expansion scheme based on an expanding (fixed output length) PRF PRF_s.

Definition of FExp**.** For a fixed expanding PRF $\mathsf{PRF}_s : \mathcal{K} \times \{0,1\}^{2n} \rightarrow \{0,1\}^{sn}$ with $s \geq 2$, FExp takes in a key $K \in \mathcal{K}$, the auxiliary info $\gamma \in \{0,1\}^{2n}$ and the

desired output length $\ell \in \mathbb{N}$ as inputs. It then uses the PRF_s as shown in Fig. 3b and outputs the key-stream $K_{exp} \in \{0,1\}^{\ell}$.
We give a formal statement of the FExp security and support it with a security proof. The formal security claim is stated in Theorem 3.

Theorem 3 (Security of FExp). *Let PRF_s be an expanding pseudorandom function with a secret and uniform random key $K \in \mathcal{K}$ and $s \geq 2$. Then for any adversary $\mathcal{A}$ who makes at most q FExp queries such that the total number of PRF_s calls induced by all the queries is at most $\sigma = \sum_{i=1}^{q} \ell_i$ with ℓ_i being the output length (in sn-bit blocks) of i^{th} query, we have*

$$\mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF}_s}(\mathcal{B}) + \frac{2q(\sigma - q)}{2^{2n}}$$

for some adversary $\mathcal{B}$ who makes at most σ queries, and runs in time given by the running time of $\mathcal{A}$ plus $\gamma_0 \cdot \sigma$ for some constant γ_0.

We use combinatorics and reduction to prove Theorem 3 and defer the proof to App. C.2.

Game **gexp-real**	Game **gexp-ideal**
$//K \xleftarrow{\$} \{0,1\}^k$	$//K \xleftarrow{\$} \{0,1\}^k$
Oracle $\mathcal{E}(\gamma, \ell)$	**Oracle** $\mathcal{E}(\gamma, \ell)$
return $\Pi(K, \gamma, \ell)$	**return** $\mathsf{RF}_{K,\gamma}[1 \ldots \ell]$
$b \leftarrow \mathcal{A}^{\mathcal{E}}$	$b \leftarrow \mathcal{A}^{\mathcal{E}}$
return b	**return** b

Fig. 7: Games **gexp-real** and **gexp-ideal** defining the security of an Exp scheme Π. Here $\mathsf{RF}_{K,\gamma}$ is a function (independently sampled for every (K,γ)) that outputs arbitrary many uniform random bits.

9 Security Analysis of Skye

The security of a KDF scheme depends on the properties of the source of initial key material (IKM, Def. 1) from which Z is chosen. We refer the reader to [28] for various examples. Although the definition there does not specify the inputs to the Σ algorithm, it provides a pair $(Z, \mathcal{C}_Z)$ where Z (the sample set) represents the secret IKM, and $\mathcal{C}_Z$ is a set of some auxiliary knowledge about Z (or its distribution). This auxiliary information is available to the attacker and can be used in the KDF security analysis. In other words, a KDF is "secure" on inputs Z even when the value $\mathcal{C}_Z$ is available to the attacker. A Diffie-Hellman value Z will consist of the value g^{xy} while $\mathcal{C}_Z$ could represent the set $\{p, q, g, g^x, g^y\}$. In a different application, say a random number generator that works by hashing samples of system events in a computer, the value $\mathcal{C}_Z$ may include some of the sampled events used to generate Z.

Towards defining the security of Skye, we recall the min-entropy and m-entropy source definitions from [28].

Definition 6 (min-entropy [28]). *A probability distribution $\mathcal{X}$ has min-entropy (at least) m if for all a in the support of $\mathcal{X}$ and for a random variable X drawn according to $\mathcal{X}$,* $\Pr(X = a) \leq 2^{-m}$.

Definition 7 (m-entropy source [28]). *We say that Σ is a statistical m-entropy source if for all s and a in the support of the distribution Σ, the conditional probability* $\Pr(Z = s|\mathcal{C}_Z = a)$ *induced by Σ is at most 2^{-m}. We say that Σ is a computational m-entropy source (or simply an m-entropy source) if there is a statistical m-entropy source Σ' that is computationally indistinguishable from Σ.*

CCS **Security.** We follow the original CCS definition from [16, 28] for a secure KDF with an m-entropy source Σ in the adaptive chosen context information (γ) model with a single salt. The CCS formalization for an m-entropy source makes use of the **ccs-real** and **ccs-ideal** security games (a simplified version is provided in Fig. 8 in App. A). The CCS-security is then defined as the indistinguishability of the KDF generated outputs K_{kdf} from truly random strings under a chosen input attack. More precisely, given a KDF scheme Π and an adaptive adversary $\mathcal{A}$, who interacts with either **ccs-real** or **ccs-ideal**, the $\mathcal{A}$'s advantage at breaking the CCS security of Π is defined as $\mathbf{Adv}_{\Pi}^{\mathsf{CCS}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\mathbf{ccs\text{-}real}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{ccs\text{-}ideal}} \Rightarrow 1]\right|$.

We now give the $\mathsf{Skye}[\mathsf{lsb}, \mathsf{PRF}_s]$ security statement in Theorem 4.

Theorem 4 (Security of Skye). *Let* $\mathsf{PRF}_s : \mathcal{K} \times \{0,1\}^{2n} \to \{0,1\}^{sn}$ *be an expanding pseudorandom function with $s \geq 2$ and key size $k \leq 212$ bits. Let $\Sigma = (Z, \mathcal{C}_Z)$ be an input source. Let Z is sampled from a set containing secret, random, independent and fresh* X3DH *handshake outputs computed over the group G, where G is defined as the cyclic subgroup of* Curve25519 *using the base point $x = 9$ [11]. Then, for all adversaries $\mathcal{A}$ who make q* Skye *queries in at most $\sigma = \sum_{i=1}^{q} \ell_i$* PRF_s *calls with ℓ_i being the output length (in sn-bit blocks) of i^{th} query, we have*

$$\mathbf{Adv}_{\mathsf{Skye}[\mathsf{PRF}_s]}^{\mathsf{CCS}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{PRF}_s}^{\mathsf{prf}}(\mathcal{B}) + \mathbf{Adv}_{G}^{\mathsf{ecddh}}(\mathcal{C}) + \frac{q}{2^{128}} + \frac{2q(\sigma - q)}{2^{2n}}$$

for adversaries $\mathcal{B}$ and $\mathcal{C}$ making at most σ and $4q$ PRF_s *and* ECDDH *(over G) queries, respectively and running in time given by the running time of $\mathcal{A}$ plus $\gamma_0 \cdot \sigma$ for some constant γ_0.*

We defer the proof of Theorem 4 to App. C.3. In our implementation, we instantiate Skye with $\mathsf{PRF}_s = \mathsf{Butterknife}$ to achieve 128-bit security i.e. $n = k = 128$.

10 Discussion and Future Research

Curve448 and 224-bit security. Signal with Skye targets 128-bit security with Curve25519. The 224-bit secure version of Signal is based on Curve448. This puts forward the question of whether the Skye syntax and similar security proof arguments apply for the 224-bit version. The general study of the underlying components of Skye is directly applicable for constructing 224-bit secure Skye. A 224-bit secure Skye can be defined with the same syntax, and similar security arguments when Curve25519 is replaced by Curve448. However, we avoid a separate security treatment of Skye with Curve448 given that no PRF with key security > 128 bits, input size > 256 bits and no idealized assumptions exists. We leave the research of finding efficient expanding PRFs with at least 224 bits security in the standard model as an open problem.

Network latency and IoT devices. The changes we propose in this work improve performance of the Signal cryptographic component, but the impact on the overall performance of modern smartphones and laptops might reduce when network latency is included. Nonetheless, this tradeoff differs for all various of devices and network setups. Low-performance hardware, e.g., IoT devices or wireless sensor networks, could greatly benefit from our improvements, enabling a wider deployment of the Signal Protocol.

Energy efficiency. Although we present improved time efficiency, an associated benefit is the lower energy consumption. Energy savings are relevant even in settings where the network latency overshadows the performance.

Security under compromised X3DH samples. To avoid the RO-model, our present Skye analysis in Signal assumes at least 3 uncompromised X3DH keys. That results in stronger standard model security when compared to HKDF. In the case when there are at most 2 uncompromised X3DH keys (and assuming the user doesn't update the compromised ones), security can be maintained albeit by modeling the PRF_s as a RO (similarly to the treatment in [20]), a subject to separate analysis.

Skye applications beyond Signal and DH sources. Our performance results indicate that similar applications, such as WhatsApp, Facebook Messenger, Skype, Allo, Status, Secure Chat, Viber, Forsta and Blockchain-based-X3DH [41] for IoTs benefit in a similar fashion when instantiated with Skye. We leave their concrete performance analysis to future research.

Note that Skye is based on the generic DExt_f, hence its security is not limited to Curve25519 points or DH sources, in general. Realizing the potential of Skye beyond DH sources, and finding corresponding weak extractors fs, is also an important direction for future works. This includes exploring its application in the present Message Layer Security (MLS) [3] and Post-Quantum Signal projects such as [13] where KDF calls are made to extract uniformly random keys from shared secrets.

On instantiation of PRF_s. In this work we use the concrete instantiation of ButterKnife. Yet, our design can accommodate any PRF_s. For example, ForkSkinny is also a suitable candidate but then the gains, when compared with ButterKnife, would be lessened. The PRF security reduces to half, 64-bits, and the performance as well, due to the fact that ForkSkinny performs slower than ButterKnife and does not have NI support on regular platforms.

DExt beyond linear systems. The number of extracted bits from DExt and the security might be improved for the same inputs when non-linear multivariate equations (with the extra cost of field multiplications) are used. Studying such systems is out of the scope and motivation of this paper and we leave the design and analysis of further generalized and efficient DExt-like extensions as an open problem.

References

1. Messenger secret conversations: Technical whitepaper `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`
2. Signal Protocol software libraries Github [accessed on 05/02/2023], `https://github.com/signalapp/`
3. The Messaging Layer Security (MLS) Protocol Work in Progress, Internet-Draft, draft-ietf-mls-protocol-17, 19 December 2022, `https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-17`
4. Alwen, J., Coretti, S., Dodis, Y.: The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 129–158. Springer International Publishing, Cham (2019)
5. Andrea Piana, Pedro Pombeiro, C.P.O.T.D.E.: Specifications for Status clients - 5/SECURE-TRANSPORT `https://specs.status.im/spec/5`
6. Andreeva, E., Bhati, A.S., Preneel, B., Vizár, D.: 1, 2, 3, Fork: Counter Mode Variants based on a Generalized Forkcipher. IACR Trans. Symmetric Cryptol. **2021**(3), 1–35 (2021)
7. Andreeva, E., Cogliati, B., Lallemand, V., Minier, M., Purnal, A., Roy, A.: Masked Iterate-Fork-Iterate: A new Design Paradigm for Tweakable Expanding Pseudorandom Function. Cryptology ePrint Archive (2022)
8. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 153–182. Springer (2019)
9. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Annual International Cryptology Conference (CRYPTO). pp. 123–153. Springer (2016)
10. Bernstein, D.J.: Cryptographic competitions: CAESAR. `http://competitions.cr.yp.to`
11. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: International Workshop on Public Key Cryptography. pp. 207–228. Springer (2006)
12. Boneh, D.: The Decision Diffie-Hellman problem. In: Buhler, J.P. (ed.) Algorithmic Number Theory. pp. 48–63. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)

13. Brendel, J., Fiedler, R., Günther, F., Janson, C., Stebila, D.: Post-quantum asynchronous deniable key exchange and the signal handshake. In: Public-Key Cryptography–PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part II. pp. 3–34. Springer (2022)
14. Chen, L., Moody, D., Regenscheid, A., Randall, K.: SP800-186, Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. Tech. rep., National Institute of Standards and Technology (2019)
15. Chevalier, C., Fouque, P.A., Pointcheval, D., Zimmer, S.: Optimal Randomness Extraction from a Diffie-Hellman Element. EUROCRYPT 2009 p. 572 (2009)
16. Chuah, C.W., Dawson, E., Simpson, L.: Key derivation function: the SCKDF scheme. In: IFIP International Information Security Conference. pp. 125–138. Springer (2013)
17. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: A security analysis of Deoxys and its internal tweakable block ciphers. IACR Transactions on Symmetric Cryptology pp. 73–107 (2017)
18. Ciss, A.A.: Two-sources randomness extractors for elliptic curves. arXiv preprint arXiv:1404.2226 (2014)
19. Ciss, A.A., Sow, D.: Two-Source Randomness Extractors for Elliptic Curves for Authenticated Key Exchange. In: International Conference on Codes, Cryptology, and Information Security. pp. 85–95. Springer (2017)
20. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A Formal Security Analysis of the Signal Messaging Protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS P). pp. 451–466 (2017). https://doi.org/10.1109/EuroSP.2017.27
21. Derbez, P., Iwata, T., Sun, L., Sun, S., Todo, Y., Wang, H., Wang, M.: Cryptanalysis of AES-PRF and its dual. IACR Transactions on Symmetric Cryptology **2018**(2) (2018)
22. Dodis, Y., Elbaz, A., Oliveira, R., Raz, R.: Improved randomness extraction from two independent sources. In: Approximation, randomization, and combinatorial optimization. Algorithms and techniques, pp. 334–344. Springer (2004)
23. Fouque, P., Pointcheval, D., Stern, J., Zimmer, S.: Hardness of Distinguishing the MSB or LSB of Secret Keys in Diffie-Hellman Schemes. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Automata, Languages and Programming, 33rd International Colloquium, ICALP, Venice, Italy, 2006, Part II. Lecture Notes in Computer Science, vol. 4052, pp. 240–251. Springer (2006). https://doi.org/10.1007/11787006_21, https://doi.org/10.1007/11787006_21
24. Gilbert, E.N.: A comparison of signalling alphabets. The Bell system technical journal **31**(3), 504–522 (1952)
25. Grassl, M.: Bounds on the minimum distance of linear codes and quantum codes. Online available at http://www.codetables.de (2007), accessed on 2021-06-25
26. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Submission to CAESAR : Deoxys v1.41 (October 2016), http://competitions.cr.yp.to/round3/deoxysv141.pdf
27. Kolyang, D., Sow, D., Ciss, A.A., Tchapgnouo, H.B.: Two-sources randomness extractors in finite fields and in elliptic curves. African Journal of Research in Computer Science and Applied Mathematics **24** (2017)
28. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Annual Cryptology Conference. pp. 631–648. Springer (2010)
29. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication (1997)

30. Liu, Y., Shi, B., Gu, D., Zhao, F., Li, W., Liu, Z.: Improved meet-in-the-middle attacks on reduced-round Deoxys-BC-256. The Computer Journal **63**(12), 1859–1870 (2020)
31. Lund, J.: Signal partners with Microsoft to bring end-to-end encryption to Skype `https://signal.org/blog/skype-partnership/`
32. Madden, N.: What's the Curve25519 clamping all about? `https://neilmadden.blog/2020/05/28/whats-the-curve25519-clamping-all-about/`
33. Marlinspike, M.: Open whisper systems partners with Google on end-to-end encryption for Allo `https://signal.org/blog/allo/`
34. Marlinspike, M.: WhatsApp's Signal Protocol integration is now complete `https://signal.org/blog/whatsapp-complete/`
35. Marlinspike, M., Perrin, T.: The double Ratchet algorithm, November 2016 `https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf`
36. Marlinspike, M., Perrin, T.: The X3DH key agreement protocol. Open Whisper Systems (2016), `https://signal.org/docs/specifications/x3dh/x3dh.pdf`
37. Maurer, U., Pietrzak, K., Renner, R.: Indistinguishability amplification. In: Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27. pp. 130–149. Springer (2007)
38. Mennink, B., Neves, S.: Optimal PRFs from blockcipher designs. IACR Transactions on Symmetric Cryptology pp. 228–252 (2017)
39. Perrin, T.: The Noise protocol framework (2016), `noiseprotocol.org`
40. Rogaway, P.: Evaluation of some blockcipher modes of operation. Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan (2011), `https://crossbowerbt.github.io/docs/crypto/rogaway_modes.pdf`
41. Ruggeri, A., Celesti, A., Fazio, M., Galletta, A., Villari, M.: BCB-X3DH: a Blockchain Based Improved Version of the Extended Triple Diffie-Hellman Protocol. In: 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). pp. 73–78 (2020). `https://doi.org/10.1109/TPS-ISA50397.2020.00020`
42. Tchapgnouo, H.B., Ciss, A.A.: Multi-sources Randomness Extraction over Finite Fields and Elliptic Curve. arXiv preprint arXiv:1502.00433 (2015)
43. Varshamov, R.R.: Estimate of the number of signals in error correcting codes. Docklady Akad. Nauk, SSSR **117**, 739–741 (1957)
44. Zhao, B., Dong, X., Jia, K.: New related-tweakey boomerang and rectangle attacks on Deoxys-BC including BDT effect. IACR Transactions on Symmetric Cryptology pp. 121–151 (2019)

A CCS Security Games

Game ccs-real

$//I_\gamma = \{\}$
$//\gamma_c = \perp$ until provided by $\mathcal{A}$
$(Z, \mathcal{C}_Z) \leftarrow \Sigma$ $//m$-entropy source

Oracle $\mathsf{Derive}(\gamma, \ell)$
define $I_\gamma = I_\gamma \cup \{\gamma\}$
if $\gamma \neq \gamma_c$
  return $(\Pi(Z, \gamma, \ell), \mathsf{salt})$
else //challenge query
  return $(\Pi(Z, \gamma, \ell), \mathsf{salt})$

$(\gamma_c, \ell_c), b \leftarrow \mathcal{A}^{\mathsf{Derive}}$
//challenge query and the
//final output bit of $\mathcal{A}$
if $\gamma_c \in I_\gamma$
  return $\perp$
else
  return b

Game ccs-ideal

$//I_\gamma = \{\}$
$//\gamma_c = \perp$ until provided by $\mathcal{A}$
$(Z, \mathcal{C}_Z) \leftarrow \Sigma$ $//m$-entropy source

Oracle $\mathsf{Derive}(\gamma, \ell)$
define $I_\gamma = I_\gamma \cup \{\gamma\}$
if $\gamma \neq \gamma_c$
  return $(\Pi(Z, \gamma, \ell), \mathsf{salt})$
else //challenge query
  return $(\mathsf{RF}_{Z,\gamma}[1 \ldots \ell], \mathsf{salt})$

$(\gamma_c, \ell_c), b \leftarrow \mathcal{A}^{\mathsf{Derive}}$
//challenge query and the
//final output bit of $\mathcal{A}$
if $\gamma_c \in I_\gamma$
  return $\perp$
else
  return b

Fig. 8: Games **ccs-real** and **ccs-ideal** defining the CCS-security of a KDF scheme. Here $\mathsf{RF}_{Z,\gamma}$ is a function (independently sampled for every (Z, γ)) that outputs arbitrary many uniform random bits and salt denotes the internally sampled salt value which is used in the queries $\mathsf{Derive}(\cdot, \cdot)$ (Note that this salt here can be a uniform random string or fixed to some constant or null string. However, we let its sampling remain general in the security definition).

B (Sub-)Optimality Analysis of DExt_f

In this section we provide the security analysis of our proposed extractor and its instantiation. We start with the basic definitions that are required for the analyses presented in this section.

Definition 8 (Consistent system). *A system of linear equations over a field $\mathbb{F}$ is called consistent if it has at least one solution in $\mathbb{F}$.*

Definition 9 (Coefficient matrix). *For a given set S of linear equations, the coefficient matrix C_S of S is defined by a matrix with i^{th} row as the coefficients of the variables in the i^{th} linear equation of the set S.*

Definition 10 (Binary linear code). *A binary linear code of length n and rank k is a linear subspace C with dimension k of the vector space $\mathbb{F}_2^k$ where $\mathbb{F}_2$ is the binary field. The vectors in C are called codewords.*

Definition 11 (Hamming distance). *For any binary linear code C, the Hamming distance between any two codewords in C is defined as the number of elements in which the codewords differ.*

Any binary linear code C can be represented by $[n, k, d]_2$ where n is the length of a codeword in C, d is the minimum of Hamming distances of all pairs of codewords in C and k is the rank of C. The maximum number of codewords in any $[n, k, d]_2$ code is denoted by $A_2(n, d)$. This implies that $k \leq \log_2(A_2(n, d))$.

We now introduce two new definitions called degree of involvement and min-degree of involvement for the upcoming analysis.

Definition 12 (Degree of involvement). *Let $S = \{E_i | 1 \leq i \leq a\}$ be a consistent system of equations of v variables over a field $\mathbb{F}$ and let x_0 be one of these v variables then the degree of involvement di_S of x_0 is defined as the minimum number of other variables on which the value of x_0 depends in S.*

To exemplify, consider the following system of one equation with 3 variables over $\mathbb{R}$, $S_1 = \{x + y + z = 3\}$ where $V = \{x, y, z\}$. Clearly, any variable in V has 2 degrees of involvement as its value depends on the other two variables. Now, as another example, consider the following system of three equations over $\mathbb{R}$, $S_2 = \{x+y+z = 3, x+y = 2, z = 1\}$. Unlike to the previous example, in S_2 where z is now fixed by another equation, we have zero degree of involvement left in the system for z. Similarly, for x and y, the degree of involvement is 1 as they depend on each other.

Definition 13 (Min-degree of involvement). *Let $S = \{E_i | 1 \leq i \leq a\}$ be a consistent system of equations of v variables defined by the set $V = \{V_1, V_2, \ldots, V_v\}$ over a field $\mathbb{F}$ then the min-degree of involvement (MDI) mdi of S is defined as the minimum of $\mathsf{di}_S(V_i)$ for $1 \leq i \leq a$.*

We note that the MDI of a system is not equal to its degree of freedom. In fact, one can show that the degree of freedom for a system S is equal to the "maximum" of $\mathsf{di}_S(x)$ over all variables x in S. However, for this work the important extremum on these degrees is the minimum defined above as the MDI of a system. The definition of degree of freedom is provided below for completeness.

Definition 14 (Degree of freedom). *Let $S = \{E_i | 1 \le i \le a\}$ be a system of linear equations of v variables defined by the set $V = \{V_1, V_2, \ldots, V_v\}$ over a field $\mathbb{F}$ and let $V_f \subset V$ be the largest subset of V such that for all possible values of the variables of V_f in $\mathbb{F}$, all equations of S holds. We use the term "free variable" to denote a variable in V_f and the degree of freedom* df *of S is defined as $\mathsf{df}(S) = |V_f|$.*

Next, we state necessary theorems towards the construction of DExt_f. We emphasize that to extract optimal randomness from a given input $a = \{a_i\}_{i=1}^{v}$ and c, one would need to find one of the largest system S of linear equations containing v variables (each variable corresponding to an element in the set a) with binary coefficients and with $\mathsf{mdi}(S) = c - 1$. Below in Theorem 5 we show that this problem is equivalent to finding a $[v, \log_2(A_2(v,c)), c]_2$ optimal binary linear code which has been considered hard for general values of v, c and that there is no polynomial-time algorithm that can find a $[v, \log_2(A_2(v,c)), c]_2$ code for arbitrary values of v, c.

Theorem 5. *Let $\mathcal{S}_c$ be the collection of all consistent systems $S^j = \{E_i^j | 1 \le i \le a_j\}$ of linear equations in v variables defined by the set $V = \{V_1, V_2, \ldots, V_v\}$ over the binary field $\mathbb{F}_{2^w}$ (for some integer $w \ge 0$) with binary coefficients and for each $S \in \mathcal{S}_c$ we have $\mathsf{mdi}(S) = c - 1$ for some positive integer $c \le v$ then for a system $S^* \in \mathcal{S}_c$ such that $|S^*| = \max_j \{a_j | S_j \in \mathcal{S}_c\}$ we have*

$$|S^*| = \log_2(A_2(v,c))\,.$$

Proof of Theorem 5 is straightforward from the fact that the coefficient matrix C_S for any system $S \in \mathcal{S}_c$ can be equivalently seen as a basis set of codewords a.k.a. the generator matrix for a $[v, |S|, c]_2$ code. Hence, for S^* as defined, we have $2^{|S^*|} = A_2(v,c)$ and thus the result of the theorem.

We further note that there exists a good lower bound on $A_2(v,c)$ called the Gilbert–Varshamov bound [24, 43] (which states that $\log_2(A_2(v,c)) \ge \lfloor v - \log_2 \sum_{i=0}^{c-2} \binom{v}{i} \rfloor$ and proves the existence of a $[v, |S|, c]_2$ code with $|S| \ge \lfloor v - \log_2 \sum_{i=0}^{c-2} \binom{v}{i} \rfloor$), however, there does not exist any deterministic method that can construct a linear code satisfying the Gilbert–Varshamov (GV) bound.

For our work, we settle with a comparatively loose lower bound of $|S| = b = \lfloor (v-c)/\lceil c/2 \rceil \rfloor + 1$ (which is very close to GV or even optimal for settings where v is very small and quite loose, otherwise) but with a deterministic algorithm $\mathcal{E}$ that can construct linear codes satisfying this bound (as shown below in Def. 15 and Theorem 6).

Definition 15. *$\mathcal{E}$ is a deterministic algorithm that, when provided with two integers v and $c \le v$, first computes a coefficient matrix C_S with its ij^{th} entry $C_S[ij]$ defined as*

$$C_S[ij] = \begin{cases} 1 & \text{if } \lceil c/2 \rceil (i-1) + 1 \le j \le \lceil c/2 \rceil (i-1) + c \\ 0 & o/w \end{cases}$$

and then for a variable set $\{V_1, V_2, \ldots, V_v\}$ *with independent variables over the binary field* $\mathbb{F}_{2^w}$ *(for some integer* $w \geq 0$*), it returns the corresponding consistent system* S *of* C_S*.*

Theorem 6. *Every system* S *returned by the algorithm* $\mathcal{E}$ *as defined above in* Def. 15 *has* $\mathsf{mdi}(S) = c - 1$*.*

We defer the proof of Theorem 6 to App. C.4 and provide Table 3 in App. D showing the differences between $\log_2(A_2(v, c))$ and $b = \lfloor (v - c)/\lceil c/2 \rceil \rfloor + 1$ for various values of v and c up to sample sizes $v \leq 10$. From the table, one can infer that our definition of b can be considered very good for small values of v and even optimal for $v \leq 5$ (which is pretty sufficient in practice).

Further, we note that the construction of DExt_f allows variable sample sizes, security parameters and output sizes, therefore, for a deterministic and faster execution of this algorithm, we fix b to $\lfloor (v-c)/\lceil c/2 \rceil \rfloor + 1$. However, we recommend that for applications where sample size, security parameter and output size is defined only once, one may use a better or even optimal code (i.e. $\log_2(A_2(v, c))$ instead of b), if exists to extract more randomness with almost the same security. We refer the reader to `codetables.de` [25] for existing tables of optimal $[v, \log_2(A_2(v, c)), c]_2$ codes for certain v, c settings.

In this paper, b is treated as a positive integer, defined as $\lfloor (v - c)/\lceil c/2 \rceil \rfloor + 1$ for another positive integers v, c. Therefore, all results that are defined in terms of b and c in this paper are directly applicable to any application for which a different and larger value for b exists.

For simplicity, we have provided (and used in the rest of this paper) an equivalent definition of the algorithm $\mathcal{E}$ as a function called invXOR (see Def. 5).

Equivalence between the two definitions can be easily understood from the fact that for a given variable set $\{V_1, V_2, \ldots, V_v\}$ and their corresponding values as a set $\{a_1, a_2, \ldots a_v\}$, each concatenation in the output of invXOR function corresponds to an equation in the generated system S of $\mathcal{E}$ and vice versa.

C Omitted Security Proofs

C.1 Proof of Theorem 1

Proof. We are given v independent z-bit elements $Z_1, \ldots, Z_v$ (when represented as binary strings) that are chosen from some public finite sets $S_1, S_2, \ldots, S_v$, respectively for some positive integer z and a public function f (that may or may not require a random salt value for its evaluation) with a range in $\{0, 1\}^w$ for some positive integer w. Let us now consider U_w as a random variable uniformly distributed over $\{0, 1\}^w$. For some $\epsilon > 0$, we have $\mathsf{SD}(f(U_{Z_i}, \mathsf{salt}), U_w) \leq \epsilon$ for all i. Since the value of salt is sampled once and used for all values of Z_i, for simplicity, we denote $f(U_{Z_i}, \mathsf{salt})$ by $f(U_{Z_i})$ in the rest of the proof. Let c, k be two positive integers and $b = \lfloor (v - c)/\lceil c/2 \rceil \rfloor + 1$ such that $w = \lceil k/b \rceil$. Now, from the definition of SD and ϵ we have

$$\mathsf{SD}(f(U_{Z_i}), U_w) = \frac{1}{2} \sum_{x \in U_w} \left| \Pr[f(U_{Z_i}) = x] - \frac{1}{2^w} \right| \leq \epsilon \,. \qquad (1)$$

Also, since an element of U_D of size k that corresponds to $Z_1, \ldots, Z_v$ can be equivalently defined as $\mathsf{invXOR}(f(Z_1)\|f(Z_2)\|\cdots\|f(Z_v), c)[1\ldots k]$, we have for $k' = b\lceil k/b \rceil$,

$$\begin{aligned}\mathsf{SD}(U_D, U_k) &\leq \mathsf{SD}(\mathsf{invXOR}(f(U_{Z_1})\|f(U_{Z_2})\|\cdots \\ &\quad \cdots\|f(U_{Z_v}), c), U_{k'}) \\ &= \mathsf{SD}(\oplus_{i=1}^{c} f(U_{Z_i})\| \oplus_{i=1+\lceil c/2\rceil}^{i=c+\lceil c/2\rceil} f(U_{Z_i})\|\cdots \\ &\quad \cdots\| \oplus_{i=1+(b-1)\lceil c/2\rceil}^{i=c+(b-1)\lceil c/2\rceil} f(U_{Z_i}), U_{k'})\end{aligned}$$

We denote $\oplus_{i=1+(j-1)\lceil c/2\rceil}^{i=c+(j-1)\lceil c/2\rceil} f(U_{Z_i})$ by U_{f_j}. Hence, we have

$$\begin{aligned}\mathsf{SD}(U_D, U_k) &\leq \mathsf{SD}(U_{f_1}\|U_{f_2}\|\cdots\|U_{f_b}, U_{k'}) \\ &= \frac{1}{2}\sum_{x\in U_{k'}} |\Pr[U_{f_1}\|U_{f_2}\|\cdots\|U_{f_b} = x] - \Pr[U_{k'} = x]| \\ &= \frac{1}{2}\sum_{\substack{x\in U_{k'} \\ x_1,\ldots,x_b \xleftarrow{w} x}} \left|\prod_{j=1}^{b}\Pr[U_{f_j} = x_j] - \frac{1}{2^{k'}}\right| .\end{aligned}$$

The last equality holds due to the fact that for each U_{f_j} the corresponding subset of Z_is contains at least one new/fresh independent element from the corresponding main set Z (note that this is true for all positive values of c). Hence all U_{f_j}s can be considered independent from each other. Now, with some basic algebra we can show that for all y_js

$$\prod_{j=1}^{b} y_j - \frac{1}{2^{k'}} = \sum_{j=1}^{b}\left[\left(\frac{1}{2^w}\right)^{j-1}\left(y_j - \frac{1}{2^w}\right)\prod_{j'=j+1}^{b} y_{j'}\right] .$$

Hence, we have $\mathsf{SD}(U_D, U_k)$

$$\begin{aligned}&\leq \frac{1}{2}\sum_{\substack{x\in U_{k'} \\ x_1,\ldots,x_b \xleftarrow{w} x}}\sum_{j=1}^{b}\left|\left(\frac{1}{2^w}\right)^{j-1}\left(\Pr[U_{f_j} = x_j] - \frac{1}{2^w}\right)\prod_{j'=j+1}^{b}\Pr[U_{f_{j'}} = x_{j'}]\right| \\ &= \sum_{j=1}^{b}\frac{1}{2^{wj-w+1}}\sum_{\substack{x\in U_{k'} \\ x_1,\ldots,x_b \xleftarrow{w} x}}\left|\left(\Pr[U_{f_j} = x_j] - \frac{1}{2^w}\right)\prod_{j'=j+1}^{b}\Pr[U_{f_{j'}} = x_{j'}]\right| \\ &= \sum_{j=1}^{b}\frac{1}{2^{wj-w+1}}\left(\sum_{x_j\in U_w}\left|\Pr[U_{f_j} = x_j] - \frac{1}{2^w}\right|\right) \\ &\qquad \left(\prod_{j'=j+1}^{b}\sum_{x_{j'}\in U_w}\Pr[U_{f_{j'}} = x_{j'}]\right)\left(\prod_{j'=1}^{j-1}\sum_{x_{j'}\in U_w} 1\right)\end{aligned}$$

$$= \sum_{j=1}^{b} \mathsf{SD}(U_{f_j}, U_w)\,. \tag{2}$$

Claim (1). For $\epsilon > 0$ defined as above we have $\mathsf{SD}(U_{f_j}, U_w) \leq (2\epsilon)^c/2$ for all $1 \leq j \leq b$.

Let us assume for a moment that Claim (1) holds then combining this result with Eqn. 2 gives us the result of Theorem 1 and completes its proof. We now prove the result of Claim (1). Let us recall that $U_{f_j} = \oplus_{i=1+(j-1)\lceil c/2\rceil}^{i=c+(j-1)\lceil c/2\rceil} f(U_{Z_i})$ for all $1 \leq j \leq b$. Again, for simplicity, we denote $1+(j-1)\lceil c/2\rceil$ and $c+(j-1)\lceil c/2\rceil$ by α and β, respectively. This implies

$$\begin{aligned}
&\mathsf{SD}(U_{f_j}, U_w)\\
&\quad = \mathsf{SD}(\oplus_{i=\alpha}^{\beta} f(U_{Z_i}), U_w)\\
&\quad = \frac{1}{2} \sum_{x_j \in U_w} \left| \Pr[\oplus_{i=\alpha}^{\beta} f(U_{Z_i}) = x_j] - \frac{1}{2^w} \right|\\
&\quad = \frac{1}{2} \sum_{x_j \in U_w} \left| \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j,\\ x_{ij} \in f(U_{Z_i})}} \Pr[\wedge_{i=\alpha}^{\beta} (f(U_{Z_i}) = x_{ij})] - \frac{1}{2^w} \right|.
\end{aligned}$$

As we know, the system of linear equations that corresponds to the outputs of invXOR (i.e. a system with equations defined by the concatenated blocks of an invXOR output) has $\mathsf{mdi} = c - 1$, which means every variable $f(U_{Z_i})$ in the equation $E_j := \oplus_{i=\alpha}^{\beta} f(U_{Z_i}) = x_j$ has at least $c-1$ degree of involvement. Now, since for all js, E_j contains exactly $\beta - \alpha = c - 1$ variables, we have for all E_js and x_{ij} as defined above

$$\Pr[\wedge_{i=\alpha}^{\beta} (f(U_{Z_i}) = x_{ij})] = \prod_{i=\alpha}^{\beta} \Pr[f(U_{Z_i}) = x_{ij}]\,.$$

Hence, we get $\mathsf{SD}(U_{f_j}, U_w)$

$$\begin{aligned}
&= \frac{1}{2} \sum_{x_j \in U_w} \left| \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j,\\ x_{ij} \in f(U_{Z_i})}} \prod_{i=\alpha}^{\beta} \Pr[f(U_{Z_i}) = x_{ij}] - \frac{1}{2^w} \right|\\
&= \frac{1}{2} \sum_{x_j \in U_w} \left| \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j,\\ x_{ij} \in f(U_{Z_i})}} \prod_{i=\alpha}^{\beta} \Pr[f(U_{Z_i}) = x_{ij}] - \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j,\\ x_{ij} \in U_w}} \prod_{i=\alpha}^{\beta} \frac{1}{2^w} \right|
\end{aligned}$$

$$\leq \frac{1}{2} \sum_{x_j \in U_w} \left| \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j, \\ x_{ij} \in U_w}} \prod_{i=\alpha}^{\beta} \Pr[f(U_{Z_i}) = x_{ij}] - \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j, \\ x_{ij} \in U_w}} \prod_{i=\alpha}^{\beta} \frac{1}{2^w} \right|$$

$$\leq \frac{1}{2} \sum_{x_j \in U_w} \sum_{\substack{\oplus_{i=\alpha}^{\beta} x_{ij} = x_j, \\ x_{ij} \in U_w}} \prod_{i=\alpha}^{\beta} \left| \Pr[f(U_{Z_i}) = x_{ij}] - \frac{1}{2^w} \right|$$

$$= \frac{1}{2} \sum_{x_{ij} \in U_w} \prod_{i=\alpha}^{\beta} \left| \Pr[f(U_{Z_i}) = x_{ij}] - \frac{1}{2^w} \right|$$

$$= \frac{1}{2} \prod_{i=\alpha}^{\beta} 2 \cdot \mathsf{SD}(f(U_{Z_i}), U_w) \leq \frac{1}{2} (2\epsilon)^c .$$

Here the first inequality holds because $|f(U_{Z_i})| \leq |U_w| = 2^w$ for all $1 \leq i \leq v$ and the last inequality follows from Eqn. 1.

C.2 Proof of Theorem 3

Proof. We first replace $\mathsf{PRF}_s(K, \cdot)$ with a uniformly sampled random function $f(\cdot) \xleftarrow{\$} \text{Func}(2n, sn))$ and let $\mathsf{FExp}[f]$ denote the FExp mode that uses f instead of PRF_s, which yields

$$\mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}[\mathsf{PRF}_s]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF}_s}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}[f]}(\mathcal{A}) .$$

Let us consider that $\mathcal{A}$ makes at most q FExp queries with i^{th} query containing ℓ_i f calls and hence calling f for total $\sigma = \sum_{i=1}^{q} \ell_i$ many times. Clearly, by definition of f, we know that all the output bits are random and uniformly distributed as long all the queried σ many inputs to f are unique. In other words, if all the queried f inputs in query i are denoted by the ordered multiset $Q_i = \{x_j^i\}_{j=1}^{\ell_i} = \{\gamma^i, K_1^i \| K_2^i, K_1^i \| (K_2^i \oplus \langle 1 \rangle), \ldots, K_1^i \| (K_2^i \oplus \langle \ell_i - 2 \rangle)\}$ then we have

$$\mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}[f]}(\mathcal{A}) \leq \Pr[\exists (i,j) < (i',j') \text{ such that } x_j^i = x_{j'}^{i'}] . \tag{3}$$

Case Analysis.
Case 1 [When $j = j' = 1$]. Under this case, all $(x_j^i, x_{j'}^{i'})$ pairs are defined as $(\gamma^i, \gamma^{i'})$ with $\gamma^i \neq \gamma^{i'}$ $\forall i \neq i'$ and hence $\Pr[x_j^i = x_{j'}^{i'}] = 0$.
Case 2 [When $j = 1 \vee j' = 1$ but $j \neq j'$]. Under this case, we always have either $x_j^i = K_1^i \| (K_2^i \oplus \langle j \rangle)$ or $x_{j'}^{i'} = K_1^{i'} \| (K_2^{i'} \oplus \langle j' \rangle)$ and since each one of $K_1^i, K_2^i, K_1^{i'}$ and $K_2^{i'}$ are outputs of a uniform random function (f), they are uniformly distributed over $\{0,1\}^n$. Therefore, $\Pr[x_j^i = x_{j'}^{i'}] = 1/2^{2n}$. W.l.o.g., let us assume that $j = 1$ and then there are total q and at most $\sigma - q$ many choices to pick (i,j) and $(i'j')$, respectively.
Case 3 [When $i = i'$ and $j \neq 1 \wedge j' \neq 1$]. Under this case, all $(x_j^i, x_{j'}^{i'})$ pairs are defined as $(K_1^i \| (K_2^i \oplus \langle j \rangle, K_1^i \| (K_2^i \oplus \langle j' \rangle)$ with $j \neq j'$ hence $\Pr[x_j^i = x_{j'}^{i'}] = 0$.

Case 4 [When $i \neq i'$ and $j \neq 1 \wedge j' \neq 1$]. Under this case, we always have $x_j^i = K_1^i \| (K_2^i \oplus \langle j \rangle)$ and $x_{j'}^{i'} = K_1^{i'} \| (K_2^{i'} \oplus \langle j' \rangle)$. Hence, we can write

$$\Pr[x_j^i = x_{j'}^{i'}] = \Pr[(K_1^i \oplus K_1^{i'}) \| (K_2^i \oplus K_2^{i'}) = 0^n \| (\langle j \rangle \oplus \langle j' \rangle)] \,. \tag{4}$$

Now, since each one of $K_1^i, K_2^i, K_1^{i'}$ and $K_2^{i'}$ are outputs of a uniform random function (f), they are uniformly distributed over $\{0,1\}^n$ and hence $\Pr[x_j^i = x_{j'}^{i'}] = 1/2^{2n}$. Additionally, we observe from Eqn. 4 with $\langle c \rangle = \langle j \rangle \oplus \langle j' \rangle$ that there are total $\sigma - q$ and at most q many choices for (i, c) and i', respectively.

Combining all these results into Eqn. 3 gives us

$$\mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}[f]}(\mathcal{A}) \leq \frac{2q(\sigma - q)}{2^{2n}}$$

and hence the result of Theorem 3.

C.3 Proof of Theorem 4

Proof. Let us first define an event E which says that the key fed to the underlying FExp component of the Skye construction over a total of q queries is secret and indistinguishable from a uniform random binary string. Let us now recall from Sec. 8.1 that if the key to the FExp construction is secret and indistinguishable from a uniform random binary string then the outputs of FExp are independent and indistinguishable from uniform random binary strings (of same length) with adversarial advantage as defined in Theorem 3. Further, one can also note from the security definition of CCS that the only difference between the real and ideal CCS games w.r.t. Skye is the corresponding outputs being uniform random or not. This as described above under the event E is upper bounded by $\mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}[\mathsf{PRF}_s]}(\mathcal{A}')$ for some adversary $\mathcal{A}'$ against FExp that uses $\mathcal{A}$ (restricted under the event E) as a subroutine. Now, for Skye with a source group G defined as in Theorem 4 (over $\mathsf{Curve25519}$) we have that under the ECDDH assumption on G, $\Pr(\neg E)$ is upper bounded by $q \cdot \mathsf{SD}(\mathsf{DExt}_{\mathsf{lsb}}(U_Z), U_k)$ and thus

$$\begin{aligned}\mathbf{Adv}^{\mathsf{CCS}}_{\mathsf{Skye}[\mathsf{PRF}_s]}(\mathcal{A}) &\leq \mathbf{Adv}^{\mathsf{ecddh}}_{G}(\mathcal{C}) + q \cdot \mathsf{SD}(\mathsf{DExt}_{\mathsf{lsb}}(U_Z), U_k) \\ &\quad + \mathbf{Adv}^{\mathsf{gexp}}_{\mathsf{FExp}[\mathsf{PRF}_s]}(\mathcal{A}') \\ &\leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF}_s}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{ecddh}}_{G}(\mathcal{C}) + \frac{q}{2^{128}} + \\ &\quad + \frac{2q(\sigma - q)}{2^{2n}} \,.\end{aligned}$$

The second inequality above is derived from Theorem 3 and Corollary 1 which states that for $k \leq 212$, $\mathsf{DExt}_{\mathsf{lsb}}$ is a $(U_Z, 2^{-128})$-deterministic extractor. This completes the proof of Theorem 4.

C.4 Proof of Theorem 6

Proof. Let S denote a system returned by the algorithm $\mathcal{E}$ as defined in Def. 15 and let C_S denote the corresponding coefficient matrix of S then we have that the ij^{th} entry of C_S can be defined as

$$C_S[ij] = \begin{cases} 1 & \text{if } \lceil c/2 \rceil(i-1)+1 \leq j \leq \lceil c/2 \rceil(i-1)+c \\ 0 & \text{o/w} \end{cases}.$$

Let $U = \{E_{i_1}, E_{i_2}, \ldots E_{i_x}\}$ denote an arbitrary subset of the system S with size $x > 1$. Clearly, if we show that the combined XOR of all equations in U always contains at least c many 1s then we can say that no linear combination of equations in S can have degree of involvement $< c-1$ and hence the claim of the Theorem.

Now, to prove the above statement, we use the following simple approach. Let us first define an indexed set U' as the sorted version of U where the equations are sorted by the value of their corresponding first column indices js in C_S with $C[ij]$ entry as 1. In other words, the sorted set U' will have the entries in the same order as they are defined in S. Clearly, the combined XOR of U will be same as of U'. Now, one can note that in this definition of C_S, every row contains at least $\lceil c/2 \rceil$ many unique 1s entries than others. Hence, the combined XOR of U' will always have the unique 1s entries of the first and the last equation of U' which in total will be $2\lceil c/2 \rceil \geq c$ many 1s.

D A Code Difference Table

c \ v	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2		0	0	0	0	0	0	0	0	0
3			0	0	0	1	1	1	1	2
4				0	0	0	1	1	1	1
5					0	0	0	0	0	1
6						0	0	0	0	0
7							0	0	0	0
8								0	0	0
9									0	0
10										0

Table 3: A table showing the differences between the actual value of $\log_2(A_2(v,c))$ and b for various values of v and c upto sample sizes $v \leq 10$. The optimal values of $\log_2(A_2(v,c))$ are taken from `codetables.de` [25].

E PRF-PRNG vs CCS security

Alwen et al. [4] proposed a syntax called PRF-PRNG for KDFs in Signal. In simpler words, a PRF-PRNG takes three inputs - the current state/key σ, an input I (which in Signal's context will be $Z\|\gamma$) and the output length ℓ (left implicit in the original description of [4]) and returns a string R of length ℓ bits and a new state/key to be updated σ'. Any PRF-PRNG = (P-init, P-up) is consist of two algorithms - P-init which is used to initiate the PRF-PRNG by generating the first state/key σ (for this it may use some preshared secret key) and P-up which is used over the PRF-PRNG inputs as defined above to generate the outputs.

The paper also defines a security notion P for PRF-PRNG schemes which we will refer here as $\mathsf{prf\text{-}prng}$ for ease of understanding. For Signal's application, the same paper mentions two KDF examples with $\mathsf{prf\text{-}prng}$ security, namely, HKDF and PRP-then-PRG [4]. We note here that HKDF is $\mathsf{prf\text{-}prng}$-secure when its underlying compression function is assumed as RO whereas the proposed PRP-then-PRG is $\mathsf{prf\text{-}prng}$-secure when the PRP is initialized with a preshared "uniform" random secret key. This assumption of preshared symmetric key can not be met in practice for two arbitrary parties that are communicating for the first time which makes PRP-then-PRG inapplicable to Signal. We also note that this incompatibility can be countered by assuming the PRP to be an ideal cipher (which is analogous to the RO assumption) and hence initializing the key with some public constant.

How to define a PRF-PRNG using KDFs with standard syntax. We first formally define a simple way of converting a KDF with the standard syntax (as defined in [28]) into a PRF-PRNG and then state the formal claim on relation among the two mentioned KDF security notions in Theorem 7.

Let $\Pi(Z,\gamma,\ell)$ and $\Pi'(Z',\gamma',\ell')$ be two standard syntax KDF functions then we can define a PRF-PRNG as Π_{PP} = (P-init, P-up) where P-init is just a constant function that returns $\sigma_0 = 0^s$ for s being the state size of the PRF-PRNG and for the $i(\geq 1)^{th}$ query to the PRF-PRNG, we have

$$\text{P-up}(\sigma_{i-1}, I_i = Z_i\|\gamma_i, \ell_i) = \begin{cases} \sigma_1\|R_1 = \Pi(Z_1,\gamma_1,\ell_1+s) & \text{when } i=1 \\ \sigma_i\|R_i = \Pi'(\sigma_{i-1}, Z_i\|\gamma_i, \ell_i+s) & \text{when } i>1\,. \end{cases} \tag{5}$$

Note that Π and Π' can be same but we don't fix them here for generality.

Theorem 7 (CCS **implies** prf-prng). *Let Π and Π' be two* CCS*-secure schemes w.r.t. distributions $\Sigma = (Z, \mathcal{C}_Z)$ and $\Sigma = (\Pi(Z,\gamma,\ell), \mathcal{C}_Z\|\gamma\|\ell)$, respectively then the PRF-PRNG Π_{PP} defined using Π and Π' as shown above will be* prf-prng*-secure under same input sources. More concretely, for all adversaries $\mathcal{A}$ who make total q Π_{PP} queries, we have*

$$\mathbf{Adv}^{\mathsf{prf\text{-}prng}}_{\Pi_{\mathsf{PP}}[\Pi,\Pi']}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{CCS}}_{\Pi}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{CCS}}_{\Pi'}(\mathcal{C})$$

for some adversaries $\mathcal{B}$ and $\mathcal{C}$ making at most q_1 and q_2 queries to Π and Π', respectively such that $q_1 + q_2 = q$ and running in time given by the running time of $\mathcal{A}$ plus $\alpha_0 \cdot q$ for some constant α_0.

Proof (Theorem 7). Let us recall from Sec. 9 and Fig. 8 that the CCS security of a KDF(Z,γ,ℓ) w.r.t. the input source $(Z, \mathcal{C}_Z)$ implies that the output of a KDF query where the input has either unique γ value or independently sampled Z value is indistinguishable from $\mathsf{RF}_{Z,\gamma}[1\ldots\ell]$ i.e. uniform random binary string of length ℓ bits. Let us now slightly abuse the notation for simplicity and denote by f and f' two functions that take inputs of the form (Z,γ,ℓ) and return $\mathsf{RF}_{Z,\gamma\|\Pi}[1\ldots\ell]$ and $\mathsf{RF}_{Z,\gamma\|\Pi'}[1\ldots\ell]$ as outputs, respectively. This gives us

$$\mathbf{Adv}^{\mathsf{prf\text{-}prng}}_{\Pi_{\mathsf{PP}}[\Pi,\Pi']}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{CCS}}_{\Pi}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{CCS}}_{\Pi'}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{prf\text{-}prng}}_{\Pi_{\mathsf{PP}}[f,f']}(\mathcal{A})\,.$$

Note that for any input $(\sigma, I = Z\|\gamma, \ell)$, $\Pi_{\mathsf{PP}}[f,f']$ as per Eqn. 5 will always return independently sampled uniform random strings $(\sigma'\|R)$ when I is unique (which implies that all returned chall-prf outputs in prf-prng games [4, Fig. 7] are indistinguishable from random strings). Similarly, for any input $(\sigma, I = Z\|\gamma, \ell)$, $\Pi_{\mathsf{PP}}[f,f']$ will always return independently sampled uniform random strings R when σ is the uncompromised current state (which implies that all returned chall-prng outputs in prf-prng games [4, Fig. 7] are indistinguishable from random strings). This implies $\mathbf{Adv}^{\mathsf{prf\text{-}prng}}_{\Pi_{\mathsf{PP}}[f,f']}(\mathcal{A}) = 0$ and hence the result. □

Clearly for Signal, setting $\Pi = \mathsf{Skye}$ (referring to KDF1 calls) and $\Pi' = \mathsf{FExp}$ (referring to KDF2 and KDF3 calls; which is CCS-secure as the input samples contain the current state value which is uniformly random and secret and thus can be used as the key to the PRF) gives us a Π_{PP} that covers all three types of KDF calls in Signal. The security here can be deduced from Theorem 7 that says for all adversaries $\mathcal{A}$ making a total of q queries to Π_{PP}, there exists some adversary $\mathcal{B}$ making at most q queries and running in time given by the running time of $\mathcal{A}$ plus some constant $\alpha \cdot q$ such that

$$\mathbf{Adv}^{\mathsf{prf\text{-}prng}}_{\Pi_{\mathsf{PP}}[\mathsf{Skye},\mathsf{FExp}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{CCS}}_{\mathsf{Skye}}(\mathcal{B})\,.$$

We emphasize that as motivated in Sec. 6, this idea of using FExp in place of full Skye for KDF2 and KDF3 calls in Signal gives significant performance benefits.

F Performance Details

In this section, we provide the benchmark tables Table 4 and 5 that correspond to the performance plots of Fig. 5a and 5b, respectively.

n	HKDF	Skye	HKDF	Skye	HKDF	Skye
	with AES-NI with SHA-NI		without AES-NI without SHA-NI		with AES-NI without SHA-NI	
1	2.22 ± 0.02	1.38 ± 0.01	6.51 ± 0.32	3.44 ± 0.03	6.47 ± 0.05	2.32 ± 0.03
2	4.48 ± 0.03	2.79 ± 0.02	12.97 ± 0.09	6.88 ± 0.04	12.95 ± 0.1	4.65 ± 0.03
3	6.7 ± 0.04	4.13 ± 0.03	19.43 ± 0.09	10.33 ± 0.22	19.4 ± 0.1	6.97 ± 0.04
4	8.92 ± 0.04	5.5 ± 0.08	25.93 ± 0.08	13.81 ± 0.49	25.88 ± 0.21	9.31 ± 0.06
5	11.14 ± 0.07	6.94 ± 0.05	32.48 ± 0.23	17.22 ± 0.21	32.36 ± 0.24	11.63 ± 0.09
6	13.34 ± 0.12	8.25 ± 0.05	38.82 ± 0.1	20.68 ± 0.05	38.86 ± 0.26	13.92 ± 0.27
7	15.68 ± 0.09	9.6 ± 0.08	45.42 ± 0.13	24.12 ± 0.46	45.33 ± 0.32	16.25 ± 0.12
8	17.98 ± 0.13	10.98 ± 0.1	51.82 ± 0.19	27.49 ± 0.07	51.86 ± 0.18	18.59 ± 0.14
9	20.07 ± 0.12	12.36 ± 0.12	58.38 ± 0.22	30.9 ± 0.27	58.35 ± 0.34	20.89 ± 0.07
10	22.33 ± 0.07	13.76 ± 0.31	64.89 ± 0.22	34.33 ± 0.29	64.81 ± 0.23	23.17 ± 0.12
20	44.44 ± 0.15	27.66 ± 1.38	129.82 ± 0.43	68.73 ± 0.18	129.48 ± 0.97	46.33 ± 0.18
30	66.68 ± 0.19	41.24 ± 0.28	194.63 ± 0.48	103.02 ± 0.58	194.16 ± 0.78	69.37 ± 0.43
40	88.66 ± 0.75	55.99 ± 0.25	260.27 ± 0.79	137.27 ± 0.68	258.92 ± 0.82	92.74 ± 0.31
50	111.6 ± 0.38	68.66 ± 0.53	324.22 ± 1.16	171.74 ± 1.59	323.76 ± 3.69	115.77 ± 0.71
60	133.79 ± 0.47	83.87 ± 0.51	389.55 ± 0.65	206.8 ± 0.52	388.39 ± 3.2	138.9 ± 1.01
70	155.49 ± 1.12	96.82 ± 0.58	453.72 ± 0.89	240.95 ± 0.64	453.18 ± 1.42	162.21 ± 1.07
80	178.02 ± 1.46	110.22 ± 0.81	518.02 ± 1.93	275.16 ± 1.84	517.8 ± 3.09	185.4 ± 0.66
90	201.36 ± 0.84	124.52 ± 0.47	582.89 ± 1.93	309.23 ± 1.77	582.69 ± 1.21	208.51 ± 1.31
100	222.8 ± 1.59	137.57 ± 1.05	649.14 ± 2.18	343.14 ± 1.99	646.73 ± 7.1	232.95 ± 10.46

Table 4: The mean time ± standard deviation (in µs) required to encrypt n messages sent by one party to the other.

n	HKDF	Skye	HKDF	Skye	HKDF	Skye
	with AES-NI with SHA-NI		without AES-NI without SHA-NI		with AES-NI without SHA-NI	
1	200.65 ± 1.12	194.51 ± 1.01	225.74 ± 1.86	204.82 ± 1.57	226.35 ± 1.37	201.42 ± 0.51
2	212.71 ± 1.81	203.19 ± 0.74	254.0 ± 0.8	221.24 ± 0.49	254.8 ± 2.01	214.08 ± 1.63
3	224.6 ± 1.79	211.19 ± 1.71	283.92 ± 7.51	238.06 ± 0.62	283.95 ± 1.05	226.12 ± 2.16
4	236.46 ± 0.69	219.88 ± 1.68	312.14 ± 0.9	254.73 ± 1.19	313.06 ± 1.02	238.73 ± 1.81
5	249.48 ± 1.52	229.85 ± 1.61	342.48 ± 2.38	272.02 ± 0.69	342.95 ± 1.1	251.37 ± 0.6
6	261.41 ± 1.79	237.91 ± 0.77	371.38 ± 1.48	288.83 ± 0.96	372.77 ± 8.9	264.32 ± 1.9
7	273.36 ± 1.4	246.89 ± 2.11	400.75 ± 1.27	305.93 ± 2.02	401.4 ± 1.54	276.52 ± 1.01
8	285.22 ± 1.13	255.93 ± 2.28	429.51 ± 1.06	322.96 ± 0.68	430.53 ± 1.11	288.91 ± 7.07
9	296.93 ± 1.6	264.1 ± 3.08	459.52 ± 1.13	337.91 ± 1.05	460.04 ± 3.28	300.99 ± 0.77
10	308.73 ± 0.64	272.63 ± 1.63	488.46 ± 1.15	355.45 ± 1.33	488.52 ± 1.71	313.48 ± 0.73
20	428.98 ± 0.89	361.04 ± 2.63	781.15 ± 6.48	522.62 ± 1.46	780.11 ± 2.08	437.46 ± 1.35
30	545.92 ± 1.08	446.9 ± 2.44	1071.67 ± 3.4	688.91 ± 3.74	1071.74 ± 9.22	559.56 ± 1.3
40	667.88 ± 8.06	535.84 ± 3.14	1364.68 ± 2.87	856.07 ± 2.04	1363.87 ± 8.33	684.44 ± 2.26
50	786.88 ± 1.78	621.98 ± 3.3	1652.43 ± 2.88	1021.6 ± 2.34	1655.0 ± 10.11	807.02 ± 2.37
60	905.83 ± 4.57	710.14 ± 10.74	1948.63 ± 12.21	1190.16 ± 4.19	1946.28 ± 3.94	935.55 ± 6.16
70	1025.61 ± 3.97	796.48 ± 5.8	2237.93 ± 4.99	1357.24 ± 2.92	2235.78 ± 4.38	1056.3 ± 2.32
80	1146.73 ± 2.48	888.52 ± 3.11	2528.96 ± 6.65	1523.47 ± 10.06	2530.07 ± 15.57	1184.09 ± 3.27
90	1263.38 ± 6.85	975.91 ± 6.12	2820.42 ± 9.19	1690.03 ± 6.13	2818.72 ± 9.55	1308.08 ± 3.49
100	1382.64 ± 2.92	1063.36 ± 8.75	3115.64 ± 18.99	1855.34 ± 3.72	3109.68 ± 17.06	1433.14 ± 7.3

Table 5: The mean time ± standard deviation (in µs) required to encrypt (and decrypt) n messages that are sent (and received) by one party to (and from, respectively) the other.