NIST Internal Report
NIST IR 8214B ipd

Notes on Threshold EdDSA/Schnorr Signatures

Luís T. A. N. Brandão
Michael Davidson

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8214B.ipd

NATIONAL INSTITUTE OF
STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Internal Report
NIST IR 8214B ipd

Notes on Threshold EdDSA/Schnorr Signatures

Luís T. A. N. Brandão
Strativia

Michael Davidson
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8214B.ipd

August 2022

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Publication History

This version is the initial public draft (ipd).

How to cite this NIST Technical Series Publication

Luís T. A. N. Brandão, Michael Davidson (2022). Notes on Threshold EdDSA/Schnorr Signatures. (National Institute of Standards and Technology, Gaithersburg, MD) NIST IR 8214B ipd.
https://doi.org/10.6028/NIST.IR.8214B.ipd

NIST Author ORCID iDs

Luís T. A. N. Brandão: 0000-0002-4501-089X
Michael Davidson: 0000-0002-4862-5697

Contact Information

nistir-8214B-comments@nist.gov

Public Comment Period

August 12, 2022 – October 24, 2022

Submit Comments

Only via email: nistir-8214B-comments@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA).

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nations measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITLs responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Abstract

This report considers threshold signature schemes interchangeable with respect to the verification mechanism of the **E**dwards-Curve **D**igital **S**ignature **A**lgorithm (EdDSA). Historically, EdDSA is known as a variant of Schnorr signatures, which are well-studied and suitable for efficient thresholdization, i.e., for being computed when the private signing key is secret-shared across multiple parties. In the threshold setting, signatures remain unforgeable even if up to some threshold number of the cosigners become compromised. The report analyzes the conventional (non-threshold) EdDSA specification from Draft FIPS 186-5, reviews important security properties, with an emphasis on strong unforgeability, and distinguishes various approaches for corresponding threshold schemes. Notably, while providing better security assurances, threshold signatures can be used as drop-in replacement for conventionally produced signatures, without changing legacy code for verification of authenticity. The report identifies various challenges and questions that would benefit from more attention, are of interest for future guidance and recommendations, and may be applicable beyond EdDSA.

Keywords

Digital signatures; EdDSA; secure multi-party computation; Schnorr; threshold cryptography; threshold schemes.

Preface

This document is intended for: technicians engaged in the development of recommendations for threshold signature schemes; cryptography experts interested in providing constructive technical feedback, or in collaborating in the development of open reference material; and all those, including from academia, industry, government and the public in general, interested in future recommendations about threshold signatures.

The reference threshold approaches identified in this document are representative examples not to be construed as preferences. See NISTIR 8214A for previous context of the NIST Multi-Party Threshold Cryptography project. Feedback is welcome from the community.

Acknowledgments

The first author performed this work as a Foreign Guest Researcher (non-employee) at NIST, while under contract with Strativia. The authors thank Lily Chen, Dustin Moody and René Peralta for their feedback.

Call for Patent Claims

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

    i) under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or

    ii) without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: nistir-8214B-comments@nist.gov

Table of Contents

List of Tables

List of Figures

Executive Summary

Digital signatures, based on public-key cryptography, underpin the security of critical information systems. They support authentication and non-repudiation, and have been standardized by NIST, via the Federal Information Processing Standard (FIPS) Publication 186. Its most recent version — Draft FIPS 186-5 — specifies three signature schemes, the most recent of which is the Edwards-Curve Digital Signature Algorithm (EdDSA).

The security of signatures relies critically on the secrecy and proper use of its private signing key. In threshold cryptography, the key is split (secret-shared) across various parties, so that a signature can be produced only if a threshold number of parties agrees. In a *threshold signature scheme*, the signing takes place without the parties ever recombining the key.

For interoperability, a threshold scheme should produce signatures that, with respect to the verification operation, are interchangeable with those produced in a non-threshold (conventional) manner. This allows for a drop-in replacement of the signature generation, without changing legacy code for verification. EdDSA, being a Schnorr-style scheme, has a linearity property that is very well suited for thresholdization, once the needed secrets have been secret-shared. However, there are various ways in which to distributively achieve those secret sharings. They give rise to a diversity of threshold approaches, with various tradeoffs.

EdDSA signatures are specified as deterministic, but their determinism is not verifiable from the signature. Thus, a variant probabilistic signature can still be interchangeable with respect to EdDSA verification. Such a variant would use a randomized or hybrid (with randomness and pseudorandomness) nonce, allowing for a simpler threshold protocols.

Threshold EdDSA has a high potential for adoption, as it enables distribution of trust for signing operations and higher resistance to certain attacks. Several considerations in this report are also applicable to other NIST-approved signature schemes specified in Draft FIPS 186-5. Allowing threshold EdDSA for pre-quantum security may also provide useful experience for the exploration of threshold schemes for post-quantum primitives.

The analysis in the present report is covered in four main sections:

- **Conventional setting:** the context of the NIST specification, and the security properties of EdDSA and interchangeable Schnorr-style signature schemes.
- **Threshold approaches:** high-level summary of four types of approaches from the literature, including both deterministic and probabilistic schemes.
- **Further considerations:** various aspects of relevance in the threshold setting.
- **Conclusions:** a synthesis of the benefits of the threshold setting, with a highlight on probabilistic schemes, and a proposal for consultation with the greater community.

The main security property of interest for EdDSA signatures is strong *unforgeability*. This ensures that an adversarial client cannot produce any signature that has not been generated by the key holder. There are other properties, such as *binding*, which can be considered from the perspective of a malicious signer.

A main concern with the implementation of EdDSA is the assurance of good nonces. The inadvertent reuse of a nonce (across different messages being signed) leaks the private key. In fact, even a slight bias in the nonce allows for key-recovery, provided enough signatures are obtained. Conversely, when the nonce is pseudorandomly generated as a transformation of a persistent secret key and the message, thus avoiding a detectable bias, some side-channel attacks may enable determining the secret key. The implementation of a hybrid mode, using both randomness and pseudorandomness, has the potential to improve on each of the two non-hybrid modes. This hybrid approach can also be useful in the threshold setting, where there are more opportunities and challenges about randomness and determinism.

There are known solutions for threshold EdDSA/Schnorr-style schemes, including distributed key generation. Recently there has been a surge of new approaches, focused on features like low number of rounds and/or simulatability, for both deterministic and probabilistic signing.

For deterministic signing, a secure multi-party computation can distributively generate a secret-sharing of a pseudorandom nonce, based on the message and a secret-shared nonce-derivation key. Another approach is to let each party provide a deterministic nonce contribution, while proving correctness with a zero-knowledge proof.

For probabilistic signing, the distributed generation of a randomized nonce can take advantage of homomorphic properties already innate to the EdDSA/Schnorr scheme. Here, it is important to safeguard security under concurrent executions, where an adversary has a view of the intermediate state of many signing operations. Recent proposals have focused on protocols with reduced number of rounds of interaction, with two and three being the norm (assuming broadcast is possible in a single round), depending on the security formulation.

There are two main frameworks used in practice to formulate and prove threshold security:

- *simulation-based* (useful for modularity and composability): where the notion of security is incorporated into an ideal functionality.
- *game-based*: where a game defines each property of interest, e.g., unforgeability.

Some considerations are inherent to the threshold setting: agreement on what to sign, malicious "random" contributions, interface between requester and cosigners, authenticated channel, timing assumptions, precomputation before receiving signature requests, failure modes, good vs. bad randomness, modularity and composability. The options related to these considerations create a diverse space of solutions that should be considered.

This document explains the potential benefits of the threshold setting. In particular, there are various advantages for probabilistic approaches. Yet, safely realizing the promise of the threshold approach requires a thorough analysis. This can be pursued with an open consultation with the community of experts, via a public call for threshold schemes, to create a testbed, gathering security formulations, technical explanations, and reference implementations. The clarification resulting from analyzing said reference material can then be helpful to synthesize recommendations about threshold signature schemes.

1. Introduction

A signature scheme enables generating a "digital signature" (hereafter just "signature") that assures the authenticity of a "message" (any digital datum). The scheme is based on a cryptographic private/public key-pair, such that only the private-key holder can produce signatures that are verifiably valid with respect to that public key [DH76]. In other words, a signature scheme is *unforgeable*. When the public key is certifiably bound to the identity of the private-key holder, a valid signature provides *non-repudiation*: the signer cannot credibly deny having produced said signature. These unforgeability and non-repudiation features underpin the security of many modern applications of information systems, including public-key infrastructures (PKI). For example, they are extensively used to prevent impersonation in cyberspace, establish authenticated channels between parties, enable contract signing with legal validity, and provide offline-verifiable authenticity of software.

NIST-specified signatures. As of August 2022, the Edwards-curve Digital Signature Algorithm (EdDSA) is the most recent signature scheme included by the National Institute of Standards and Technology (NIST) in a Federal Information Processing Standard (FIPS), albeit still in draft mode: Draft FIPS 186-5. This FIPS also specifies the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Rivest–Shamir–Adleman (RSA) signature schemes. Both EdDSA and ECDSA, relying on the infeasibility of computing discrete logarithms (and related assumptions) over approved elliptic curves, allow signatures noticeably shorter than RSA, which relies on the infeasibility of integer factorization (and related assumptions). For example, at an estimated level of 128 bits of security, EdDSA and ECDSA signatures have a bit length of 512, which is one-sixth of the 3072 bits required by RSA signatures.

The threshold setting. The critical reliance on signature schemes requires a careful consideration of the techniques that help ensure the secrecy of the private signing key. The multi-party "threshold" setting allows for a distribution of trust of the private key, by use of *secret sharing* [Bla79; Sha79]. The key is split (i.e., "secret shared") across multiple parties, such that no coalition of up to some *corruption threshold* number f of faulty parties is able to recover the key. Furthermore, the actual cryptographic operation of interest — in this case signing — can be performed by any quorum with a stipulated *participation threshold*. The signing takes place without reconstructing the key. Moreover, the signatures remain *unforgeable* by a coalition of up to f malicious parties, without the help of other honest parties. The study of threshold schemes has been active for over three decades [Des88; DF90]. More recently, the NIST Internal Report NISTIR 8214A proposed that a focused analysis takes place, to collect expert feedback that can be useful as a basis for developing recommendations about threshold schemes.

Schnorr and thresholdizability. EdDSA [BDLSY11; RFC 8032] is based on Schnorr signatures [Sch90], which have been subject to extensive analysis in the literature. They have the special feature of one of their components resulting from a linear combination of two secret elements: the private signing key s and the (per-message secret) nonce r. This linearity allows for simple threshold schemes based on a linear secret-sharing of the two secrets. The matter becomes more elaborate when considering the nature of the nonce: pseudorandom (deterministic) vs. randomized. The essential property is that r remains indistinguishable from random. The secret-sharing of a random nonce can be easily achieved by leveraging independent contributions from each party. Conversely, the threshold production of a pseudorandom nonce based on the EdDSA specification is considerably more complex. It requires an expensive distributed (multi-party) computation of a specific hash over a secret-shared input. Fortunately, probabilistic versions of EdDSA, when properly parameterized, are interchangeable with respect to the verification algorithm of standardized EdDSA.

EdDSA relevance. In applications where succinctness matters, RSA signatures may be too long, and those based on elliptic curves may be preferred. In a threshold context, EdDSA may be prefered to ECDSA because the process for threshold generation of interchangeable signatures is far simpler. This report discusses the properties of conventional (non-threshold) and threshold schemes interchangeable **w**ith **r**espect **t**o (w.r.t.) EdDSA verification, paving the way to possible future recommendations or guidance about the latter.

Avoiding bias. The Draft FIPS 186-5 specification of EdDSA requires the use of a pseudorandom nonce (i.e., deterministic, depending on a secret key). While this avoids the catastrophic security breakdown in case of a biased "random" nonce, it raises a concern about higher vulnerability to some side-channel attacks. Fortunately, determinism is not the only solution to the mentioned problem. By properly adding a random component, as input to the pseudorandom transformation already used by deterministic schemes, it is possible to create a probabilistic scheme that minimizes the risk of bias. The EdDSA verification algorithm works interchangeably with randomized and with deterministic signatures. In fact, determinism is not a standalone verifiable property of EdDSA signatures.

Toward guidance. After summarizing the NIST Draft FIPS 186-5 requirements of the conventional EdDSA, this document puts in perspective various aspects of interest to corresponding Schnorr-based threshold schemes. This is intended to support possible future NIST recommendations promoting secure implementations of threshold signatures interchangeable with respect to the EdDSA verification algorithm. It is worth noting that Schnorr/ EdDSA is already widely deployed and used, albeit with variations of the curves and parameters. For example, these signatures are used in Transport Layer Security (TLS), Secure Shell Protocol (SSH), Signal, The Onion Router (TOR) / Invisible Internet Project (I2P) and Domain Name Server Security Extensions (DNSSEC), as well as some cryptocurrencies.

Document organization. Section 2 explains the notation. Section 3 establishes the NIST context about the EdDSA specification, and analyzes some security properties, including its non-verifiable determinism. Section 4 compares various approaches to thresholdize EdDSA/Schnorr. Section 5 comprises additional considerations relevant to future guidelines and recommendations about threshold signatures. Section 6 concludes with a summary of insights and a recommendation for a public call for threshold signature schemes interchangeable w.r.t. the NIST specified EdDSA verification.

2. Notation

This section explains the acronyms, abbreviations and symbols used in the document.

2.1. Acronyms

Table 1. Acronyms

Acronym	**Extended form**
AES	**A**dvanced **E**ncryption **S**tandard
CA	**C**ertification **a**uthority
CSM	**C**ryptographic **S**ecurity **M**odule
CMA	**C**hosen **m**essage **a**ttack
DKG	**D**istributed **k**ey **g**eneration
DSS	**D**igital **S**ignature **S**tandard
ECC	**E**lliptic-**c**urve **c**ryptography
ECDSA	**E**lliptic-**C**urve **D**igital **S**ignature **A**lgorithm
EdDSA	**Ed**wards-curve **D**igital **S**ignature **A**lgorithm
EUF	**E**xistential **u**n**f**orgeability
FIPS	**F**ederal **I**nformation **P**rocessing **S**tandard
HMAC	**H**ash-**b**ased **m**essage **a**uthentication **c**ode
KOSK	**K**nowledge **o**f **s**ecret **k**ey (assumption)
LSS	**L**inear **s**ecret **s**haring
MPC	[Secure] **m**ulti**p**arty **c**omputation
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NISTIR	**NIST I**nternal or **I**nteragency **R**eport
NIZKPoK	**N**on-**i**nteractive **z**ero-**k**nowledge **p**roof **o**f **k**nowledge
PKCS	**P**ublic-**k**ey **c**ryptography **S**tandards
PKI	**P**ublic-**k**ey **i**nfrastructure
PVSS	**P**ublicly **v**erifiable **s**ecret **s**haring
PRF	**P**seudo**r**andom **f**unction
RFC	**R**equest **F**or **C**omments, from the Internet Engineering Task Force
RSA	**R**ivest–**S**hamir–**A**dleman (cryptosystem or signature scheme)
RSA-SSA	**RSA S**ignature **S**cheme with **A**ppendix

Table 1 (continued from previous page)

Acronym	**Extended form**
RSA-PSS	**RSA**-based **P**robabilistic **S**ignature **S**cheme
SHA	**S**ecure **H**ash **A**lgorithm
SHAKE	**SHA** combined with **KE**CCAK
SP 800	(NIST) **S**pecial **P**ublication in Computer Security
SS; SSS	**S**ecret **s**haring; **s**ecret **s**haring **s**cheme
SUF	**S**trong **unf**orgeability (or **s**trongly **unf**orgeable)
TLS	**T**ransport **L**ayer **S**ecurity (a communication protocol)
UTC	**C**oordinated **U**niversal **T**ime (a time standard)
UC	**U**niversal **c**omposability (or **u**niversally **c**omposable)
UF	**U**n**f**orgeability (or **U**n**f**orgeable), in an EUF-CMA sense
VSS	**V**erifiable **s**ecret **s**haring
ZK; ZKP	**Z**ero **k**nowledge; **z**ero-**k**nowledge **p**roof
ZKPoK	**Z**ero-**k**nowledge **p**roof **o**f **k**nowledge

2.2. Abbreviations

The report uses some abbreviations: det. (deterministic); discrete log (discrete logarithm); e.g. (*exempli gratia* = for example); i.e. (*id est* = that is); iff (if and only if); keygen (key generation); prob. (probabilistic); pub key (public key); vs. (versus); w.r.t. (with respect to).

2.3. Symbols

The symbols of some variables were chosen to match the notation used in Draft FIPS 186-5. These often vary across the literature. The colors red, blue and green are sometimes used to help identify private input or intermediate values, public output or intermediate values, and public input values, respectively. However, color identification is not required for understanding the descriptions.

2.3.1. Symbols useful for the conventional setting

Table 2. Symbols for conventional setting

Symbol	**Description**
$+, \cdot$	Binary operators for integer addition and multiplication.

Table 2 (continued from previous page)

Symbol	Description
+, **−**	Binary operators for addition and subtraction of two elliptic curve elements.
•	Non-commutative binary operator used to multiply an elliptic curve element (on the right) by an non-negative integer (on the left), e.g., $s \bullet G$.
$\xleftarrow{\$}$	Random sampling of a value.
b	Bit-length (multiple of 8) of the public key Q, and the initial private key d. EdDSA signatures σ have $2b$ bits. Approved values: 256 and 456.
c	Binary logarithm (3 for Ed25519, 2 for Ed448) of the cofactor 2^c (order of small subgroup); useful to compare cofactorless vs. cofactored verification.
χ	Challenge component computed in the `Sign` and `Verify` operations.
ctx	Context (optional parameter in some signature modes).
d	*Precursor* private key of the signature scheme. It is the hash pre-image used to derive the *signing key* s and the *nonce-derivation key* ν.
$E_{i,j}$	Some encoding function (the subscripts are used to differentiate encodings).
G	Base point (aka generator), generator of the subgroup $\mathbb{G}$ of prime order n.
$\mathbb{G}$	Subgroup generated by G. It is the domain of public keys. It is the large subgroup (or order n) of the elliptic curve group (of order $2^c \cdot n$)
H	Some cryptographic hash function (subscripts can be used to differentiate between hash functions).
κ	Standardized security level (estimated bits of strength, e.g., 128 or 224).
M	Message (string) being signed.
μ	Index identifying the mode of a signature scheme.
n	Prime order of the elliptic curve subgroup generated by G.
Q	Public key of the signature scheme, equal to $s \bullet G$.
r	Nonce (secret).
R	Commitment of the nonce r; used as the first component of the signature.
s	*Signing key* (also called *hdigest*1 in Draft FIPS 186-5): it is the 1st half of the digest of the private key d. It is used to generate the public key Q, and to compute the 2nd component (S) of each signature.
ν	*Nonce-derivation key* (*hdigest*2 in Draft FIPS 186-5): it is the 2nd half of the digest of the private key d; used to pseudorandomly generate each nonce.
S	Second component of the signature, obtained via a linear combination of the signing key s and the secret nonce r, with the help of the challenge χ.
σ	Signature — a pair (R,S) of elements.

2.3.2. Symbols specific to the threshold setting

Table 3. Symbols for threshold setting

Symbol	Description
f	Corruption threshold (smaller than t) w.r.t. unforgeability. With "mixed adversaries" one may differentiate thresholds across types of corruptions.
n	Total **n**umber of "parties" (share-holders) [does not include the requester client, coordinators and others without a share of the private key].
$\mathscr{P}$	Set of possible cosigners (aka **p**arties) — there are n of them.
$\mathscr{P}'$	Set of cosigners agreed to participate in a particular signing execution.
P_i	One of the parties (share holders) — the index i is used similarly for shares of contributions, to identify to which party they correspond.
sid	**S**ession **id**entifier (to distinguish sessions in a concurrent setting)
t	Reconstruction **t**hreshold (usually $t = f + 1$) of the baseline secret sharing.
t'	Participation **t**hreshold: minimum size of quorum needed to generate a signature, when the number of corrupted parties does not exceed f.

For simplicity we assume throughout the paper that f is also the corruption threshold for key-recovery, being equal to the corruption threshold for the underlying secret-sharing of the signing key. However, there are conceivable protocols where the corruption threshold for unforgeability is lower than that for key-recovery.

2.3.3. On the use of square brackets []

In the present document, square-bracketing is used for various purposes.

1. **Secret-sharing.** To represent a (linear or additive) secret-sharing of the enclosed element, when used in some operation, to indicate that a vector of operations takes place. For example, $[d] \cdot G$ indicates that each secret-share d_i of d is multiplied by the base point G, with each such operation being performed locally by a different party. In Draft FIPS 186-5, the use of brackets in a left-side multiplier (e.g., $[d]$) is instead used to indicate that the enclosed element is an integer, thus distinguished from the group element (on the right side) $\mathbb{G}$.
2. **Optional argument.** When nested inside a parenthesis, to indicate an optional argument of a function, e.g., $f(a,b[,c])$.
3. **Predicate evaluation.** When embracing an equality with question mark, to enclose a predicate evaluation/verification, e.g., $[x =^? y]$.

3. The conventional EdDSA and Schnorr schemes

The **Ed**wards-curve **D**igital **S**ignature **A**lgorithm (EdDSA) is a signature scheme specified in the Draft FIPS 186-5 "Digital Signature Standard (DSS)". EdDSA operates over elliptic curves, whose allowed parameters are specified in Draft SP 800-186. The NIST specification is based on RFC 8032, which in turn was based on prior work [BDLSY11; BJLSY15]. EdDSA is a variant of the Schnorr signature scheme, itself a proof of knowledge of a discrete logarithm (discrete log) [Sch90].

The EdDSA scheme specifies a triple $(\texttt{keygen}, \texttt{sign}, \texttt{verify})$ of algorithms. It operates over an elliptic curve group of known order $2^c \cdot n$, where n is prime and c is a short integer (2 or 3). However, the actual operations (in additive notation) are performed in the cyclic subgroup $\mathbb{G}$ of order n, with an agreed *base point* G, the generator. Fig. 1 shows a simplified version (missing some encoding details) of the formula for an EdDSA signature. Notably, the 2nd element (the S) of the signature is a linear combination of the signing key s and the secret nonce r, once the public challenge χ has been calculated. This linearity is a distinctive feature of Schnorr/EdDSA-style signatures, as compared to ECDSA.

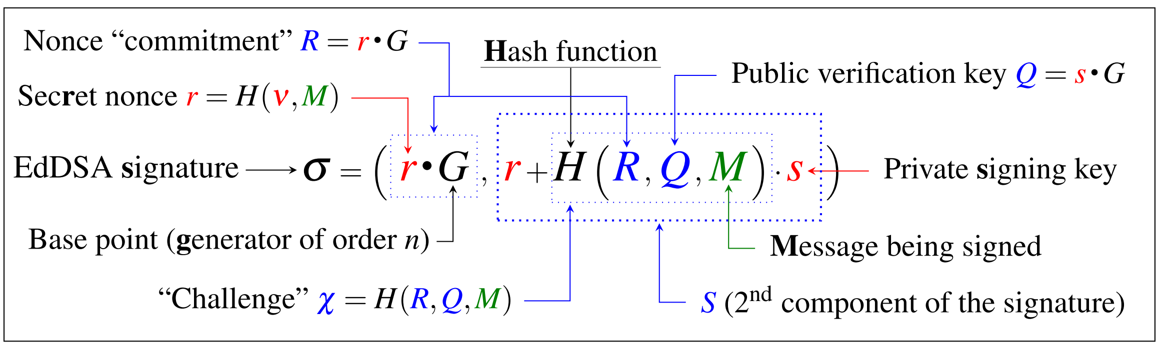

Figure 1. Annotated simplified formula of an EdDSA signature

The secrecy of the private signing key s (which is actually a cryptographic digest of the precursor private key d) depends on the infeasibility of computing "discrete logs" (in traditional multiplicative notation). In additive notation (as usual with elliptic curves, and as used in this document), this requires that it be infeasible to compute which integer s needs to multiply the base-point G to yield the public key $Q = s \cdot G$. The generation of the secret nonce r for each message requires the use of a nonce-derivation key v (which is actually another cryptographic digest of the precursor private key d), which must also remain secret. The property of unforgeability also depends on the one-wayness (or collision resistance, depending on the signature mode) of the hash function H.

EdDSA as a variant of Schnorr. The EdDSA signature of a message M can be interpreted as a (transferable) non-interactive zero-knowledge proof of knowledge (ZKPoK) of the discrete-log (the private signing key) of the public key, with the property that M is bound to the proof. The binding is done by including M in the pre-image of the ZKPoK "challenge" element χ that is determined as a hash, according to the Fiat-Shamir heuristic [FS87]. This ZKPoK approach for a signature was devised by Schnorr in 1989 [Sch90]. While the original Schnorr scheme is probabilistic, the standardized EdDSA signature (per Draft FIPS 186-5) is deterministic, since its secret nonce $r = H(v||M)$ is pseudorandom. The original Schnorr scheme includes the challenge χ in the signature, whereas EdDSA replaces it with the nonce commitment R. This change of format requires a change in the verification operation, but the rationale for unforgeability is similar, since both R and χ can be obtained from any of the signatures. More concretely: $\chi = H(R, Q, M)$ and $R = S \cdot G - \chi \cdot Q$. Based on the above, EdDSA is sometimes said to be a Schnorr-style signature, or a variant of Schnorr.

NIST-approved curves and modes. The Draft FIPS 186-5 specifies two Edwards curves (with corresponding subgroups $(\mathbb{G},+)$), for two corresponding security levels: curve Edwards25519 for 128-bit strength; curve Edwards448 for 224-bit strength. Each of the two curves allows two signing modes, w.r.t. whether the signed message is pre-hashed or not. The Draft FIPS 186-5 specifies four allowed EdDSA modes: Ed25519, Ed25519ph, Ed448, Ed448ph. The suffix "ph" means the message is prehashed when given as input to the `Sign` operation, and these modes are sometimes called HashEdDSA. The preceding part "EdXXX[XX]" identifies the underlying elliptic curve. Note that RFC 8032 defines an extra mode Ed25519ctx that is not approved in Draft FIPS 186-5. Consequently, in Draft FIPS 186-5, Ed25519 is the only mode (out of four) that does not use a context field (denoted *ctx* in Fig. 3 and Table 5).

Other curves and modes. In this document, the mode is sometimes left implicit, using a "simplified" description that omits details about the used curves, the differentiated hash functions, encodings and/or a "context" argument. The logic of EdDSA can for the most part be modularized away from these details. Thus, when some of these details are abstracted away, some of the rationale may be applicable to non-standardized parameters. For example, while the Draft FIPS 186-5 specification requires Ed22519 or Ed448 for the curve, and SHA-512 or SHAKE256 for hashing, a Schnorr variant used in Bitcoin [WNR20] specifies secp256k1 for the curve and SHA-256 for hashing. Nonetheless, when actual interchangeability with Draft FIPS 186-5 EdDSA verification is required, the focus is on the concrete standardized modes summarized in Table 5.

Pre-Quantum. EdDSA is not a post-quantum secure scheme. It is plausible that a future quantum computer will be able to use any EdDSA public verification key to determine the corresponding secret signing key. Therefore, EdDSA may in the future be decommissioned in favor of post-quantum alternatives. Nevertheless, EdDSA is currently an important signature scheme with useful features. Guidance regarding how to thresholdize it can thus be useful as a way to enable distribution of trust.

3.1. Schemes interchangeable w.r.t. EdDSA verification

NISTIR 8214A proposed the notion of interchangeability that is relevant for this document. A secure scheme is said to be *interchangeable* w.r.t. the verification algorithm of (deterministic) EdDSA signatures if the `Verify` algorithm accepts, without distinction, the variant signatures. In particular for EdDSA, this applies to a probabilistic distribution of the nonce, such as uniformly at random from $\mathbb{Z}_n$.

Figure 2 shows a simplified description of a generic signature scheme interchangeable w.r.t. EdDSA verification. It abstracts the nonce generation to fit several possibilities and omits various details deferred to Fig. 3. A probabilistic variant of EdDSA can use a random nonce. In a hybrid mode, it can also be a hash whose pre-image includes a secret key, as well as some fresh randomness per signature. See Section 3.5 for security considerations about these variants.

- `Keygen`$[n]$: { (private key) $s \leftarrow^{\$} \mathbb{Z}_n$; (public key) $Q = s \bullet G$; output (s, Q) }.
- `Sign`$[s](M)$: $\{r \leftarrow$ `GenNonce`$(\ldots)$; $R = r \bullet G$; $\chi = H(R, Q, M)$;
  $S = r + \chi \cdot s \pmod{n}$; output $\sigma = (R, S)\}$.
- `Verify`$[Q](M, \sigma)$: $\{\chi' = H(R, Q, M)$; output `accept` iff $S \bullet G =^{?} R + \chi' \bullet Q\}$

Legend: χ (challenge); G (base point, i.e., generator of $\mathbb{G}$); `GenNonce(...)` (procedure used to **gen**erate the secret nonce); M (**m**essage being signed); n (order of the group generated by G); Q (public key); r (secret nonce); R (nonce commitment; first component of the signature); s (private signing key; in the detailed scheme it is obtained as a digest — hdigest1 — of a precursor private key d); S (second component of the signature); σ (signature); $\leftarrow^{\$}$ (random sampling); $+$, $\cdot$ (integer sum and multiplication); $\mathbf{+}$, $\bullet$ (sum and multiplication-by-constant in additive group $\mathbb{G}$). Extra verification details are required.

Figure 2. (Simplified) EdDSA-style scheme, with generic nonce

Key-prefixing. The inclusion of the public key Q in the hash-calculation of the challenge χ is a best practice (known as key-prefixing) that addresses concerns w.r.t. application settings with more than one public key [Ber15; BCJZ21]. It is used in EdDSA, but it is actually not considered in the original Schnorr signature scheme [Sch90]. Hereafter in this document, the reference to "Schnorr" type signatures is considered (sometimes implicitly) only within the scope of key-prefixed versions.

Non-verifiable determinism. The EdDSA signing procedure defined in Draft FIPS 186-5 generates a deterministic signature, since `GenNonce` is a hash-based pseudorandom function. However, the deterministic property is not verifiable from the signature itself, without the secret signing key. This lack of verifiable determinism distinguishes EdDSA (and ECDSA) from some other schemes (see Table 4). Particularly, the RSA Signature Scheme with Appendix (SSA) — RSASSA-PKCS-v1_5 — part of the Public Key Cryptography Standards (PKCS) incorporated in Draft FIPS 186-5 produces verifiably deterministic signatures. (Note that Draft FIPS 186-5 also specifies an RSA-based Probabilistic Signature Scheme (PSS): RSA-PSS-PKCS-v2_1.)

At considerable computation cost compared to that of producing a signature, a signer could produce a ZKP that an EdDSA signature was correctly generated with the prescribed secret

Table 4. Determinism vs. verifiable determinism of signature schemes

Signature scheme	Is the signature algorithm deterministic?	Is the output signature verifiably deterministic?
RSASSA-PKCS	**Yes**	**Yes**
EdDSA	**Yes**	**No**
Deterministic ECDSA	**Yes**	**No**
RSA-PSS	No	No
(Probabilistic) ECDSA	No	No

nonce. Such a ZKP is outside the scope of the EdDSA specification.

3.2. Detailed EdDSA procedures

The next subsections describe the three EdDSA operations: `Keygen`, `Sign`, and `Verify`. In comparison with the simplified Fig. 2, the pseudo-code describing EdDSA in Fig. 3 includes: a parameter μ to differentiate various EdDSA modes (encoding, curves, and hash functions); details about the pseudorandom nonce generation; the use of a cofactor c in the verification mechanism; and the differentiation between signing key s and nonce-derivation key v. Table 5 gives further details for Hash and `GenNonce`.

3.2.1. Keygen

As an asymmetric-key signature scheme, EdDSA requires a private signing key s for signing, and a public verification key Q to validate signatures. As specified in Draft FIPS 186-5, the private signing key is in fact derived from a precursor private key d of the scheme. Specifically, d is hashed to yield a pair (s, v) of secret digests, which are then used separately. For simplicity, some encoding details (explained in Draft FIPS 186-5) are being omitted here, namely on how some bits in the extremities of the digests need to be preset, and on how the strings are converted into integers. The first digest — the signing key s — is used in two ways: (i) it is multiplied by the base point G to yield the public key $Q = s \cdot G$; (ii) it is used in the signing process to derive a linear form S that combines the nonce and the challenge. The second digest — the nonce-derivation key v — is used only in the signing process, to derive a message-specific secret nonce r. In practice, the two digests can be computed once in the `keygen` phase and stored, for use thereafter in the signing phase; otherwise they can be recomputed from d during each signing operation.

As described in Table 5, EdDSA has parameters approved for two security strengths (called *requested_security_strength* in Draft FIPS 186-5) κ: 128 and 224. The private key d is required to be obtained using an approved random bit generator (RBG) as a string with at least b bits. The integer b must be a multiple of 8 and is at least double κ: $b = 256$ for

`Keygen`$[b]$: {
  (private key) $d \xleftarrow{\$} \mathbb{Z}_2^b$
  $s \| v = \text{Hash}(d)$;
  (public key) $Q = s \bullet G$;
  output (d, Q) }

`Sign`$[d](\mu\, [, ctx], M)$: {
  $s \| v = \text{HashK}_\mu(d)$;
  $r = \texttt{GenNonce}[v](\mu\, [, ctx], M) \in \mathbb{Z}_n$;
  $R = r \bullet G$;
  $\chi = \text{HashC}_\mu([ctx \|] R \| Q \| f(M))$;
  $S = r + \chi \cdot s \pmod n$;
  output $\sigma = (R, S)$ }

`Verify`$[Q](\mu\, [, ctx], M, \sigma)$: {
  $(R, S) = \sigma$;
  if not $0 \le S < n$, then `reject`;
  $\chi = \text{HashC}_\mu([ctx \|] R \| Q \| f(M))$;
  $S' = 2^c \bullet S$; $R' = 2^c \bullet R$; $\chi' = 2^c \bullet \chi$;
  if $S' \bullet G =^? R' + \chi' \bullet Q$
    then output `accept`,
    else output `reject` }

Legend/notation: b (number of bits of private key, as well as of public key; it is a multiple of 8); 2^c (cofactor — 8 for Ed25519, 2 for Ed448 — needed for cofactored verification); χ (challenge); ctx (optional *context* string, empty by default, only available for the Ed25519ph, Ed448 and Ed448ph modes, i.e., not available only for the Ed25519 mode; d (private key of the signature scheme); f (transformation **f**unction applied to the message: identity for regular EdDSA; some hashing for HashEdDSA); G (base point, aka generator, of a subgroup $\mathbb{G}$ of prime order n); HashK (hash function used to derive the secret keys s and v); HashC (hash function used to derive the challenge χ); μ (mode: Ed25519, Ed448, Ed25519ph, Ed448ph, respectively encodable as (2,0), (4,0), (2,1), (4,1) — see details in Table 5); M (message being signed); q (order of $\mathbb{G}$); Q (public key, for verification); r (secret nonce); s (private signing key); v (private key for nonce generation; it is called *hdigest*2 in Draft FIPS 186-5); R (public commitment of nonce); $(+, \cdot)$ (integer sum and multiplication); $(+, \bullet)$ (sum and multiplication-by-constant in additive group $\mathbb{G}$). $=$ (assignment); $=^?$ (equality check); $||$ (concatenation). For simplicity, details about encodings are omitted. As secret input to the `Sign` algorithm, both the signing key s and nonce-derivation key v can be used instead of the precursor key d.

Figure 3. EdDSA pseudo-code and notation

$\kappa = 128$; $b = 456$ for $\kappa = 224$. Note that for $\kappa = 224$ the private key length b is 8 beyond the double, as defined in the RFC. Hereafter, d is simply assumed to be uniformly selected from $\mathbb{Z}_b = \{0, ..., 2^b - 1\}$.

3.2.2. Sign

The signing procedure (`Sign`) involves generating a pseudorandom nonce r (secret), whose procedure `GenNonce` varies with the signature mode, as described in Table 5. The "Prob" types (rows 6 and 7), although not FIPS-approved, are "interchangeable" in the sense of being verifiable as correct signatures by the FIPS-approved `Verify` algorithm. For that reason they are of interest to consider in the threshold setting, where some advantages will emerge from the use of randomness.

Table 5. EdDSA variants

Type	Standard	Mode μ	κ	$b = \|d\|$	$s\|\|v$	GenNonce r	Challenge χ
Det.	EdDSA	Ed25519	128	256	$H_0(d)$	$H_0(v\|M)$	$H_0(R\|Q\|M)$
		Ed448	224	456	$H_1(d)$	$H_1(E_{4,0}(ctx)\|v\|M)$	$H_1(E_{4,0}(ctx)\|R\|Q\|M)$
	HashEdDSA	Ed25519ph	128	256	$H_0(d)$	$H_0(E_{2,1}(ctx)\|v\|H_0(M))$	$H_0(E_{2,1}(ctx)\|R\|Q\|H_0(M))$
		Ed448ph	224	456	$H_1(d)$	$H_1(E_{4,1}(ctx)\|v\|H_2(M))$	$H_1(E_{4,1}(ctx)\|R\|Q\|H_2(M))$

Type	Variation	Mode μ	κ	$b = \|d\|$	$s\|\|v$	GenNonce r	Challenge χ
Prob.	Random	—	—	—	—	$\xleftarrow{\$} \mathbb{Z}_q$	—
	Hybrid	—	—	—	—	$H(v, rand, f(M))$	—

Legend: Some symbols are better contextualized in Fig. 3. Det. (deterministic). Prob. (probabilistic). s, v (first and second halves, respectively, of Hash(d), also denoted as 1st and 2nd digests of d; before encoding into an integer, some bits in the left and right extremities of each of these digests is preset — see details in Draft FIPS 186-5). $E_{i,j}(...)$ (encoding function, defined in FIPS 186 as dom$i$(j,...), where i is 2 or 4, corresponding to the Ed25519 or Ed448 curves, and j is 1 or 0, corresponding to whether or not it is a "pre-hash" mode). H (some cryptographic hash function or extendable output function); H_0 (SHA-512); H_1 (SHAKE256-length-912); H_2 (SHAKE256-length-512); $rand$ (secret randomness or any other secret material). The four deterministic modes (Det.) are based on Draft FIPS 186-5. The two probabilistic variants (Prob.) produce signatures interchangeable w.r.t. EdDSA verification.

The actual signature is a pair $\sigma = (R, S)$, whose first element is a "commitment" R of the secret nonce r. The second element is a linear combination $S = r + \chi \cdot s$ of the nonce r and of the first digest s of the signing key (d), applying as slope factor in the latter a hash-based "challenge" χ. The challenge χ is computed as a cryptographic hash of the commitment R, the public key Q and the message M, as shown in Table 5. Some modes (all except Ed25519) can also use a context string ctx to determine the nonce r and the challenge χ. The hash functions (and encodings) vary depending on the signature mode.

On the meaning of "commitment" in reference to R. The name "nonce commiment" given to R is used for convenience, but it should be understood in a sense more loose than that of a typical commitment scheme. The latter has two phases (commit and open), and needs to satisfy binding and semantic hiding properties. Conversely, the use of R as a "commitment" of the nonce r never requires an open phase, and its hiding property is only as provided by the application of a one-way permutation (which, being a bijection, does not semantically hide the input). The binding is satisfied unconditionally.

3.2.3. Verify

The verification procedure (`Verify`) corresponds to checking a relation between the components (S and R) of the signature, the public parameters (Q and G) and the message M. The operation requires recomputing the challenge χ, which in turn also depends on the signed message M, and then performing two multiplications and one group addition. All values (Q, R and S) are to be checked for canonical encoding. The actual verification operation specified in Draft FIPS 186-5 is called *cofactored*, as it includes a cofactor adjustment (multiplication by 2^c) of S, R and χ.

Both *cofactorless* (i.e., without cofactor adjusmtent) and *cofactored* verifications validate signatures generated per Draft FIPS 186-5 signing specification. However, cofactored verification is less strict, also validating "signatures" outside the subgroup $\mathbb{G}$, i.e., with components in a subgroup different from the one generated by G [CGN20].

It is worth noting that an additional check (not specified in Draft FIPS 186-5) on the public key Q and the nonce commitment R — namely that their order is not smaller than the cofactor 2^c — can be used to protect against some key substitution attacks [BCJZ21, Table 2].

Batch verification. In Draft FIPS 186-5, the EdDSA `Verify` algorithm is only specified for individual signatures. However, in practice some applications amortize the cost of simultaneous verification of multiple signatures (possibly across different messages and public keys). This can be done as a single verification using an adjusted S, R, and Q, with each adjusted element being obtained as the same random linear combination (i.e., with random coefficients) of the corresponding elements used across all signatures [CGN20]. An accepted test implies an overwhelming probability, in the size of the random linear coefficients (e.g., 128 bits), that all of the individual signatures would pass their respective verifications.

3.3. Strong unforgeability

Unforgeability is the essential security property of a signature scheme. It considers an adversary not knowing the private signing key s, but being able to obtain, from a signing oracle, signatures on many chosen messages [GMR88]. A scheme is "**e**xistentially **unf**orgeable against a **c**hosen **m**essage **a**ttack" (EUF-CMA) if no such adversary can produce a new valid signature (denoted *forgery*) σ for a previously unsigned message. For simplicity, this is hereafter simply referred to as UF — the existential ("E") and the CMA aspects remain implicit. The interest in this document is in a stronger notion: **strong** UF (**S**UF) [CD95, Remark 2], where the adversary cannot produce any new previously unseen message/signature pair (M,σ) that is accepted by the `Verify` algorithm. (The acronym SUF should not be confused with the notion of **s**elective **unf**orgeability, which is a notion weaker than existential unforgeability, in both the regular and strong senses). That is, SUF requires, in addition to UF, that the adversary be unable to construct an alternative signature for a message that has already been signed. More formally, SUF requires the adversary to have a negligible probability (in the security parameter κ) of winning the following game:

1. The `keygen` phase takes place as prescribed and the private key remains secret, i.e., known only to a signing oracle.
2. The adversary can choose up to q messages — $\{M_i : i = 1,...,q\}$, for which it can obtain corresponding valid signatures σ_i from the oracle.
3. The adversary wins the game if it can output a previously unseen pair (σ_{q+1}, M_{q+1}), for which $\texttt{Verify}[Q](M_{q+1}, \sigma_{q+1})$ outputs `accept`.

Note that, in the SUF game (as well as in a corresponding UF game), the adversarial capability varies between deterministic and probabilistic signatures. In the latter case the adversary receives a different signature each time it repeats a query for the signing oracle to sign the same message. The UF and SUF notions for signatures are the direct analogue of the same type of properties for message authentication codes (MAC) in the symmetric key setting [BKR00; BN08].

Strong unforgeability implies unforgeability, i.e., if a scheme is SUF, then it is also UF. This is because the adversarial goal in the SUF game is less ambitious than in the UF game. Moreover, if a scheme is *verifiably deterministic* and UF, then it is also SUF, since it is infeasible to produce more than one valid signature for the same message (as in the case of RSASSA-PKCS-v1_5; see Section 3.1). However, both probabilistic and non-verifiably deterministic schemes can be UF without being SUF.

The study of Schnorr/EdDSA unforgeability has been the subject of much research, with techniques such as the forking lemma [PS00, Theorem 4] in the programmable random oracle model, and other results ([PS96, Thm 13], [FF13; KMP16; RS21; BCJZ21]). Assuming the infeasibility of solving the "discrete log" problem in the underlying elliptic curve and the one-wayness of the hash function, the EdDSA specified in Draft FIPS 186-5 provides strong unforgeability. The HashEdDSA mode additionally requires collision resistance from the hash function.

Intuitively, SUF of EdDSA stems from SUF of Schnorr signatures, where the adversary has access to multiple random signatures for each message. The adversary in EdDSA can only get one signature per message which, although deterministic, is indistinguishable from random. Still, the details matter for an actual proof [BCJZ21]. Note that achieving SUF requires checking that the signature components are in a canonical representation. For EdDSA, this requires (as specified in Fig. 3) checking that S is a positive integer less than n. Otherwise, replacing S by $S+n$ would trivially produce a valid forgery violating SUF.

A signature scheme that is interchangeable with Draft FIPS 186-5 EdDSA verification is not automatically unforgeable. While interchangeability only depends on the `Verify` function, unforgeability also depends on the space and distribution of signatures. Consider the pathological case of a signing algorithm that always uses the same nonce even when signing different messages. Such a scheme would allow extraction of the private key when the adversary queries the signing oracle on two different messages (see Section 3.5.1), and is therefore forgeable. Other pathological examples of interchangeable schemes can be devised to break strong unforgeability without breaking UF, or break UF without allowing key-recovery (see Section 5.2.4).

3.4. Binding and non-repudiation

The classical notion of unforgeability, where the adversary is external to the signer, does not consider all possibly desirable security properties of a signature scheme. For example, SP 800-57-P1-R5 specifies that: a "Digital Signature" is "*the result of a cryptographic* *transformation of data that, when properly implemented with a supporting infrastructure and policy, provides the services of: 1. Source/identity authentication, 2. Data integrity authentication, and/or 3. Support for signer non-repudiation.*"

The unforgeability game considers the case of an adversary without knowledge of the private key. What happens, however, if the adversary controls the signer, i.e., knows and/or is able to generate the private key, and then tries to manipulate the signature generation against an unwary verifier? That may jeopardize the "data integrity authentication" requirement, even if maintaining "source/identity authentication". For example, an unforgeable signature scheme may still allow a malicious signer to produce two messages (possibly under two different public keys) and one signature that validates both messages [CGN20; BCJZ21].

3.4.1. Binding

The EdDSA verification specified in Draft FIPS 186-5 provides a form of *binding* that follows trivially from the collision resistance of the hash used to calculate the challenge χ. Considering a fixed public key Q, a malicious signer cannot find two messages M and M' and a signature σ that validates both of them under that public key. Thus, when the signer's identity is certifiably bound to a single public key Q, such as when relying on a PKI, then a signature σ binds the signer to a single message.

A stronger binding notion [CGN20; BCJZ21] goes further, considering that the public key may also be manipulated: a signature scheme provides *strong binding* if no malicious signer is able to find two different pubkey–message pairs — (Q,M) and $(Q\text{'},M\text{'})$ — and a signature σ that is valid against both pairs. In the case of EdDSA, such a collision can be obtained by a malicious signer, by using a public-key Q that is part of the small subgroup. This allows the signer to later perform a key-substitution attack: after initially sending (M,σ), w.r.t. public key Q, the signer later claims that it has actually sent $(M\text{'},\sigma)$ w.r.t. a public key Q'. While having one of the keys being in the small subgroup is not compliant with the EdDSA keygen phase, such a key is nonetheless not caught as incorrect in the standardized EdDSA verification. As already briefly mentioned in Section 3.2.3, this can be fixed by adding a simple additional verification regarding the public key Q and the nonce commitment R.

Binding can even be considered in a stronger sense, across various signature schemes and parameters (e.g., approved EdDSA and ECDSA modes), which may use different hash functions H, base-points G, encodings E_μ, moduli n and even `Verify` algorithms. For example, one can ask whether one can find a signature simultaneously valid for EdDSA and for ECDSA, each with their own parameters.

3.4.2. Non-repudiation

The colloquial expression "non-repudiation" means the inability of a signer to *repudiate* (plausibly deny) having produced a signature w.r.t. a message. However, the expression leaves some room for ambiguity, as evident by comparing the two notions explained below.

Such ambiguity can be resolved by expressing the needed non-repudiation features in terms of unforgeability and binding properties.

A (weak) notion of non-repudiation considers that the signature can be used "to support a determination by a third party of whether a message was actually signed by a given entity" [SP 800-57-P1-R5], if it can be assumed that the private key is indeed private. This property is implied by SUF, since SUF implies that any valid message–signature pair must have been created by a holder of the private key. Even if a SUF scheme is non-binding in the sense of allowing a malicious signer to produce, under the same public key, two messages and one signature that validates both messages, it still follows that both messages must indeed have been signed by the entity that knows the private key. EdDSA, being SUF, provides non-repudiation in the mentioned sense.

Some application settings may warrant a stronger notion of "non-repudiation", equivalent to binding. The following is an example application setting where a false repudiation occurs despite of the use of a SUF signature scheme. Consider, hypothetically, a non-binding signature scheme used in an application where an honest signer, upon request by a server A, generates and sends to A two messages M_0 and M_1, and a corresponding single signature σ that validates both messages. Later, the signer is asked to securely send to another server B one of those messages, M_b, for some b of the client's choice. If server B is unaware of the non-binding property, it may think that the authenticity of the message sent by the client is protected by the accompanying signature σ. However, if server A controls the communication channel, it could now replace the message by M_{1-b}, without the client or the server B realizing it, even though server B could check that the received signature σ is valid for the received message M_{1-b}. Alternatively, if server A is honest (and thus server B actually receives the original message M_b), then a malicious client can later plausibly *repudiate* that it sent said message, and claim that the message was in fact M_{1-b}, and that, plausibly, server A may have tampered with the communication. The use of a signature scheme with strong binding would make this repudiation implausible.

3.5. Nonce implementation issues

Even if the unforgeability of the specified EdDSA algorithms is assumed or proven (see Section 3.3), there are still potential security issues that arise from the implementation. The security of signatures interchangeable w.r.t. EdDSA verification depends critically on the secrecy and unbiased selection of the nonce r used in any signature (R,S). For example, should a nonce ever be known to an adversary, the signing key s can then be recovered, simply as $s = \chi^{-1} \cdot (S - r) (\bmod n)$. Other subtle issues within the `GenNonce` procedure can cause catastrophic security failures. The same type of issues apply to implementations of the ECDSA signature scheme, against which the mentioned attacks have been demonstrated. To summarize (also see Table 6):

- Implementations of probabilistic nonces may introduce biases, and even small biases can result in full recovery of the private signing key.

- Deterministic nonce generation prevents bias, but is more subject to side-channel and fault injection attacks that also enable key recovery.
- The upshot is that it can be more secure to generate the nonce in a *hybrid* manner, by adding some random noise to an otherwise deterministic procedure.

Table 6. Types of nonce generation

Nonce generation type	Bias attacks	Side-channel and fault injection attacks
Deterministic: Pseudorandom, based on a secret key	Not applicable	More vulnerable
Purely random: Entropy independent of secret key	Vulnerable	Less vulnerable
Hybrid: Randomness and pseudo-randomness	Not applicable	Less vulnerable

3.5.1. Nonce reuse

A serious nonce-related security failure occurred when the ECDSA signing key of a home video game console was recovered [bmss10]. This is due to the use of the same nonce when signing different pieces of software. A similar attack is possible if nonces are reused when EdDSA-signing different messages. In that case, from two signatures (R, S) and (R', S'), one can find the secret key by solving a pair of linear equations with two unknowns. From $S' - S = (r' - r) + s \cdot (\chi' - \chi) \pmod{n}$, and $r = r'$, the secret key follows as $s = (S' - S)(\chi' - \chi)^{-1} \pmod{n}$.

Nonce reuse can occur when an adversary is able to perform a "rewinding" attack. For example, if the signer is running in a virtual machine and nonces are generated before the message to be signed is determined, an adversary may rewind the virtual machine in order to obtain signatures on two different messages using the same nonce and different challenges. This attack can be prevented by generating the nonce in a way that depends on the message to be signed, as happens in the pseudorandom nonce generation specified for EdDSA in Draft FIPS 186-5. Some system models may also avoid rewinding concerns based on other assumptions on fresh randomness, such as selecting the nonce via a non-rewindable hardware random-number generator that produces true fresh randomness on every call.

3.5.2. Partial knowledge of random nonce

Partial information about nonces can be leaked through a poorly implemented or biased random number generator [BH19], as well as various side-channel attacks, such as cache-timing side-channels [ANTTY20]. Deliberately injected faults can also induce bias in the nonce [TTA18]. This bias can be leveraged to recover the private signing key by solving the Hidden Number Problem (HNP) [BV96] using one of two known techniques. Fourier analysis [Ble00; ANTTY20] is used when there is a very small bias (potentially even less than a single bit [ANTTY20]) but the adversary has access to many signatures; lattice-based techniques [HS01] can be used when the bias is more significant but the adversary has access to fewer signatures.

3.5.3. Side-channel and fault injection attacks against deterministic nonce

A pseudorandom (deterministic) nonce generation avoids the issues caused by bad randomness. However, that may result in a signing process more susceptible to side-channel [ABFJLM18] and fault injection attacks [RP17; SB18]. For example, differential power analysis on the modular addition operation within SHA-512 can enable the recovery of the nonce-derivation key v being hashed. Also, a differential fault injection attack can induce a "glitch" during the computation of the challenge χ, resulting in a faulted challenge hash χ'. Then, the computation follows with the proper formula $S = R + \chi \cdot s$, but using the incorrect challenge value χ', leading to an invalid signature component S' that is nonetheless a linear relation of the secret key and a secret nonce. Since the signature is pseudorandom, the adversary can additionally obtain a valid signature component S for the same message, using the same nonce r as before (and thus the same R as before), and necessarily having a different (correct) challenge χ. From S and S' the adversary can recover the private key, similar to as when a nonce is reused when signing different messages (see Section 3.5.1). The exploitation of these vulnerabilities often requires physical access to the signing device.

3.5.4. Hybrid nonce generation — combined randomness and determinism

The security issues mentioned above can be mitigated by using a hybrid mode of nonce generation, combining both random and pseudorandom components. As with deterministic nonce generation, the nonce can be computed as the output of a pseudorandom function (using as key the nonce-derivation key), whose input is the message. However, to protect against side-channel and fault-injection attacks, the function can additionally take some random bytes as input. The actual details on how the randomness and the nonce-derivation key are possibly intertwined when used as input to the pseudorandom function may depend on the concrete side-channel protection being sought.

Even if there is some bias in the used randomness, the use of a PRF (dependent on the secret nonce-derivation key) will prevent the bias from being apparent in the nonce itself. The idea is not new [SBBDS18; PSSLR18]. It has also been suggested as an update [MTR22] to RFC 8032 (on which the EdDSA specified in Draft FIPS 186-5 is based), which after the encoding of the nonce derivation key v would concatenate a random string (with the same length as v), used as a preimage to the hashing that computes the nonce.

Furthermore, as long as the "random" values contributed to this function do not repeat for the same message, there is some additional protection against side-channels and fault-injection attacks. With a single signer, if the needed entropy is unavailable at signing time, the signing simply falls back to the deterministic mode. (The threshold setting requires particular attention against insider attacks, as discussed in Section 4.3.1).

4. Threshold approaches

This section surveys, at a high level, several approaches for threshold signatures with potential interchangeability w.r.t. EdDSA verification. Section 4.1 provides intuition about the linear operations involved in a semi-honest probabilistic setting. Section 4.2 describes a template protocol for threshold Schnorr/EdDSA signatures, matching at a high-level many concrete protocols. Section 4.3 explains several deterministic approaches, while Section 4.4 considers probabilistic approaches.

4.1. Intuition for efficiency of threshold [probabilistic] Schnorr signatures

The baseline building block assumed available for threshold signatures is a **s**ecret **s**haring (SS) scheme . From an initial secret value x, the SS scheme allows producing a vector $[x] = \langle x_1, x_2, ..., x_n \rangle$ of shares, usually for distribution across n parties, such that any subset of t parties can reconstruct the secret x, but any subset of $t-1$ colluding parties learns "nothing" about the secret. For example, Shamir SS [Sha79] selects a random polynomial of degree $t-1$, subject to its evaluation at zero being the secret x; then the various shares are the evaluation of the polynomial at other points. The evaluation points across shares must not collide (which would affect the threshold guarantee) and must not be zero (which would reveal the secret).

For Schnorr signatures in particular, it is most useful to use a **l**inear SS (LSS) scheme. Linearity enables local computation of the sum of shares, and multiplication-by-constant of shares. Therefore: if $z = x + y$, it follows that $[z] = [x] + [y]$ (i.e., each local share z_i can be obtained as $x_i + y_i$.); also, if $z = a \cdot w$, then $[z] = [a \cdot w]$ (i.e., each local share z_i can be obtained as $a \cdot w_i$). The threshold properties of the secret-sharing $[z]$ upon these linear operations remains the same (namely t shares are required to reconstruct a secret). It should be noted that different secret-sharing schemes exist and can be useful, including those with multiplicative properties.

Compared with ECDSA, the better efficiency of threshold Schnorr signatures comes from being able to compute the signature operations (all linear) locally at each party, once the needed shares are distributed. In particular, when the nonce is allowed to be randomized (as in regular Schnorr, although not in EdDSA), then even the distributed secret selection of the nonce and the calculation of its commitment depend only on simple linear/homomorphic operations. Conversely, ECDSA requires computing the modular inverse of a secret-shared element, which is more complicated and inefficient to perform in a distributed manner. The non-linear operation requires interaction and may be based on a different type of secret sharing (e.g., multiplicative) and a corresponding final conversion to linear secret sharing.

This simplicity is captured well in a semi-honest threshold implementation (i.e., where every party behaves according to the protocol specification), as summarized in Table 7. In this case, the distributed computation only involves the secret-sharing and corresponding reconstruction of secret elements, as well as simple homomorphic operations. The description is for n-of-n signatures. The k-out-of-n case is resolved by Lagrange interpolation in the exponent, which can also be done with (homomorphically) linear operations.

Table 7. Conventional Schnorr vs. baseline semi-honest threshold Schnorr

Phase	Conventional	Semi-honest threshold baseline
Key-Gen	$Q = s \cdot G$	$[Q] = [s] \cdot G$; then open Q
Commit nonce	$R = r \cdot G$	$[R] = [r] \cdot G$; then open R
Compute challenge	$\chi = H(R, Q, M)$	*Same as in conventional*
Produce signature	$S = r + \chi \cdot s \pmod{n}$	$[S] = [r] + \chi \cdot [s] \pmod{n}$; then open S
Verify signature	$S \cdot G \stackrel{?}{=} R + \chi \cdot Q$	*Same as in conventional*

To "open" a public value (Q, R and S) means that every party reveals their corresponding share (Q_i, R_i and S_i, respectively), so that everyone can reconstruct the corresponding public value.

For each row involving secret material, the baseline threshold version simply computes the needed public element shares (Q_i, R_i and S_i) by homomorphic computations over the secret-shared secret values (s_i and r_i). Some additional care is required to deal with active/malicious adversaries, which in practice leads to some variations (e.g., how the nonce, or signature shares are produced), while leaving the compute challenge and verify signature steps identical to the conventional scheme.

On regular threshold signatures vs. multi-signatures. Sometimes it is useful to clearly distinguish between two types of distributed signature schemes:

- **(Regular) Threshold Schemes:** there is a fixed public key Q, whose corresponding private signing key s is secret-shared across various parties.
- **Multi-Signature schemes:** there is a setting where each party P_i has a public key Q_i, and a corresponding private signing key s_i, and any subset of them can come together to produce a multi-signature, which can only have been produced by a collaboration of all corresponding private keys, and whose verification is based on either (i) a list of the Q_i's of all signatories, or (ii) an aggregate public key Q that is derived from them.

The case of n-out-of-n (regular) threshold signatures has some similarities to a multi-signature from n parties. In particular, overlooking the `Keygen` phase, the `Sign` and `Verify` phases of a multi-signature scheme can be transformed into those of a n-out-of-n regular threshold scheme, by fixing the public key Q and the set of n parties. In both types, there is a threshold security property: an adversary must corrupt all n cosigners in order to forge a signature. Furthermore, Schnorr multi-signatures can be interchangeable w.r.t. the EdDSA verification algorithm, provided that the aggregate public key is given. For the most part, the discussion in this report considers threshold schemes in the regular sense. However, considering the above, it is sometimes useful to consider "threshold schemes" in a broad sense that also includes multi-signatures.

4.2. A template threshold Schnorr/EdDSA signature

A conventional signature scheme is composed of three procedures: `Keygen`, `Sign`, and `Verify`. A threshold implementation of it alters only the `Keygen` and `Sign` operations, which relate to private key. The verification operation (`Verify`) remains unchanged. Hereafter, the focus is on actively secure protocols (i.e., against malicious adversaries).

4.2.1. Key Generation

In the `Keygen` phase, each party obtains a share s_i of the private signing key s. During this process, every party also learns all "public" keys Q_i associated to the private keys of each other party, from which anyone can derive the global public key Q. The secret sharing (SS) can be one from several kinds, including **v**erifiable SS (VSS) or **p**ublicly **v**erifiable SS (PVSS), where each party learns additional information that enables verifying that their share is correctly related to the global public key.

The generation of these keys typically follows one of two main approaches:

- **centralized (by a dealer):** a dealer (trusted or untrusted) determines the private signing key s, then produces a secret sharing $[d]$ of the private signing key, and sends a different secret share d_i to each party i. The public key $Q = s \cdot G$, as well as its shares $Q_i = s_i \cdot G$, are sent to every party.
- **distributed (by the signatories):** the parties interact in a distributed key-generation protocol, such that no party knows the global secret key d. Typically, each party generates their own secret key share and corresponding public key share, which are then combined to generate the global public key.

Note: In the actual (deterministic) EdDSA there is also a nonce-derivation key v. In a threshold (deterministic) EdDSA scheme, functionally equivalent to EdDSA, the parties also obtain corresponding secret shares v_i. There is nonetheless an essential difference across the two private keys, w.r.t. the distributed signature process: for the signing key s, there are homomorphic properties that facilitate the group operations to be carried out in secret-shared mode; the same does note apply for the SHA-based hash-related operations performed on v. There are other threshold schemes interchangeable w.r.t. EdDSA verification that avoid the latter problem by deriving independent local nonce derivation keys per party, or even simply assuming access to good randomness.

Distributed key generation (DKG) approach. A DKG for public keys has a basic goal of letting each party obtain a secret-share of a random private key s. For typical discrete-log based schemes, the homomorphic properties of the group are such that an additive secret sharing $s_i \cdot G$ of the private key allows the calculation of (now in additive notation) a share $Q_i = s_i \cdot G$ of the public key Q. A useful gadget for DKG is a VSS scheme [CGMA85]. In particular, Feldman's scheme [Fel87] allows for non-interactive verifiability. After an interactive (e.g., 2 rounds of communication) secret-sharing, each share s_i "proves its own validity" via a verification algorithm that checks it against a commitment of the secret s.

An initial DKG scheme [Ped91] based on Feldman's VSS allowed a malicious party to bias the public key. While such a public key may still be sufficient for some purposes, it does not emulate the case of a random public key selected by a trusted dealer. A later protocol [GJKR99] solves that issue, by ensuring that any party must propose their contribution before they can learn the resulting public key. This can be achieved by adding an initial communication round where parties commit to their contributions, e.g., using Pedersen commitments [Ped92]. The mentioned DKG, for an honest majority setting and assuming broadcast channels, can be used as a basis for subsequent threshold Schnorr-style signing [SS01]. Other alternatives may be possible with a different number of rounds, depending on the system model.

Rogue-key attack. Some restrictions need to be enforced w.r.t. the key shares, in order to protect against "rogue key" attacks, where a malicious party sets their public key share Q_i to some function of the honest parties' public keys. For example, consider a 2-of-2 multi-signature scheme intended to prove that both Alice and Bob have participated in creating a signature. Let honest Alice have private key s_A and public key $Q_A = s_A \cdot G$. Bob, who is malicious, has private key s_B and public key $Q_B = s_B \cdot G$. Alice says her public key is Q_A, while Bob says his public key is $Q' = Q_B - Q_A$ (instead of the correct Q_B), even though Bob is unaware of the discrete log of Q'. The resulting shared public key is Q_B, so Bob can sign for the group without Alice's consent.

To prevent such attacks, each party may be required to prove **k**nowledge **o**f their **s**ecret **k**ey (KOSK), using a NIZKPoK of DL (base G) of Q_i, essentially equivalent to producing a signature with their private key. Some multi-signature schemes operate in the plain public key model, where parties are not required to prove knowledge of their secret keys in order to thwart rogue key attacks. This involves tweaking the procedure for generating an aggregated public key, as well as modifying the process for generating signature shares.

4.2.2. Signing

In each threshold signing session, the parties need to obtain *agreement* on several parameters: the message M to be signed; the set $\mathscr{P}'$ of cosigners actively participating in the signing session; and a session identifier *sid* used to distinguish between concurrent executions. Unless otherwise noted, the remainder of this section assumes there is a mechanism whereby parties agree on the tuple $(sid, \mathscr{P}', M)$. In practice, however, threshold implementations must explicitly consider this agreement.

In an actively secure threshold Schnorr signature, some variations or extra steps are required as compared to the semi-honest setting (Section 4.1). A simple template for threshold probabilistic signing [SS01] is to perform a DKG to obtain a secret-shared secret nonce r, along with each party receiving the nonce-contribution commitment R_i of everyone, and then let each party locally compute and broadcast their corresponding signature share. Some tricks can reduce the number of rounds, but special care is required to prevent the challenge χ from being maliciously manipulated in a way that could break unforgeability.

1. **Nonce commit.** Each party computes a random nonce share r_i and then the corresponding commitment $R_i = r_i \cdot G$. The details of this computation define whether the overall EdDSA implementation is deterministic (see Section 4.3), or probabilistic (see Section 4.4). Due to homomorphism, the commitment R_i is also a share of the commitment R of the random secret nonce r (of which no party is aware). In other words, the distributed system produces secret-sharings $[r]$ and $[R] = [r] \cdot G$. The shares of $[R]$ are then revealed between all parties, which allows each party to locally reconstruct the public commitment R. Special care is required to thwart attacks where an adversary tries to manipulate the challenge χ (dependent on M, R, and Q), possibly in a concurrent setting with many distributed signing operations taking place [DEFKLNS19]. This manipulation can be prevented by having a round of communication where parties, when committing to their nonce contribution (e.g., r_i), do not immediately reveal a share (e.g., $R_i = r_i \cdot G$) of the nonce commitment, or with more advanced techniques that can eliminate a round of communication. For example, the nonce commitment R may be a more complex linear combination of the shares R_i, using additional coefficients to avoid some malleability attacks. The revealing of the shares R_i of the public value R follows after a corresponding commitment phase, to ensure independence of values. A secret-sharing $[r]$ of a SHA-based pseudorandom nonce r would require a more generic (secure) **multi** **p**arty **c**omputation (MPC).
2. **Compute challenge.** In the simplest (and EdDSA-interchangeable) case, the challenge χ is locally computed by each party, as a hash of the nonce commitment R, the public key Q, and the message M. Some modes also include a context component *ctx* (see Table 5), or other small tweaks.
3. **Signature shares.** Based on the linear properties of the secret-sharing scheme, each party can locally compute a share of the output signature This can be as simple as $[S] = [r] + \chi \cdot [s]$ (in $\mathbb{Z}_n$). However, some protocols use sophisticated techniques where some of the elements may be tweaked. The final signature can then be computed by anyone collecting all signature shares.

The above description is for n-out-of-n signatures. The k-out-of-n case is resolved by additionally using Lagrange coefficients.

4.3. Deterministic threshold Schnorr

In a deterministic threshold Schnorr signature scheme, each message leads to a single possible signature, once the public key and/or the subset of signatories is fixed. In particular, the secret nonce r (i.e., the discrete-log of the nonce commitment R) is deterministic, even through never computed by a single party. It could seem that this can be trivially achieved by having each party provide a deterministic contribution $R_i = r_i \cdot G$, for a locally computed deterministic r_i. However, a careless protocol could result in a key-recovery vulnerability against internal adversaries (see §4.3.1). Therefore, a protocol needs to be carefully crafted, possibly using an MPC (see §4.3.2) or ZKP (see §4.3.3) that ensures correct behavior from the signatories. Table 8 compares various aspects of different deterministic approaches.

Table 8. Threshold approaches for deterministic signatures

Reference	Function-ally equi valent?	EdDSA Interchan-geable?	Same signature per message? Per/across quorums	Across re sharings	Some gadgets
[BST21, §5]	Yes	Yes	Yes/ Yes	Yes	MPC gadgets
[BST21, §6]	No	Yes	Yes/ Yes	Yes	MPC-friendly hash
[GKMN21]	No	Yes	Yes/ No	No	ZKGC, COT
[NRSW20]	No	Yes	Yes/ No	N/A	ZKP-friendly PRF

Some schemes implement the HashEdDSA mode (see Table 5). The last row [NRSW20] corresponds to a multi-signature scheme, for which the resharing does **n**ot **a**pply (N/A), since that would imply a change in public key. COT = **c**ommitted **o**blivious **t**ransfer. ZKGC = **ZK**Ps from **g**arbled **c**ircuits. The approaches also differ in efficiency, allowed thresholds, and cryptographic assumptions.

4.3.1. A key-recovery pitfall

Suppose the secret nonce r is a naive combination of "deterministic" nonce contributions from the various parties. Consider now two executions to sign the same message M. Since the determinism is not verifiable, a malicious party can provide different nonce contributions in both, whereas the honest participants supply the same deterministic nonce each time [MPSW19]. This allows the adversary to learn:

- two different challenges (χ, χ'), since they respectively depend on the two different nonce contributions from the malicious party;
- two different signature shares (S_i, S_i') from each honest party, since they depend on the two different challenges,

The above pairs from honest parties will both have been derived using the the same secret nonce r_i (prescribed to be deterministic) and the same secret signing share s_i. This enables the malicious party to obtain the secret key share of each honest party, by solving a simple pair of linear equations, leading to: $s_i = (\chi - \chi')^{-1} \cdot (S_i - S_i') \pmod{n}$.

Secure versions of deterministic threshold EdDSA/Schnorr need to resolve the above mentioned problem. Two such approaches are described below.

4.3.2. MPC-based threshold (deterministic) EdDSA

The above described pitfall (§4.3.1) can be avoided by directly using generic MPC to ensure that the secret nonce r is a hash whose pre-image includes the nonce-derivation key v [BST21], exactly as prescribed for (deterministic) EdDSA.

1. **KeyGen:** use a dealer or a dealerless keygen, such that each party has a secret share s_i of the signing key, and a secret share v_i of the nonce-derivation key.

2. **Nonce commit:** use generic MPC to compute a nonce-commitment $R = r \cdot G$, without anyone learning the corresponding discrete-log r (the nonce), and yet be assured that the nonce satisfies the prescribed relation, i.e., $r = \text{Hash}(v, \text{Hash}(M))$ in case of HashEdDSA. Considering the original SHA-based hash as a Boolean circuit, the techniques used to perform its distributed computation can be based on MPC *gadgets*, say, to obtain a secret-sharing $[r]$ of the nonce, which can then be homomorphically converted to the corresponding commitment shares $[R]$. The distributed hashing can be based on *garbled circuits*, and using *oblivious transfer* to handle the secret inputs of the circuit evaluator. Alternatively, the circuit evaluation can proceed by computing over bits that are secret-shared using a LSS scheme and a mechanism for authentication of shares. To convert between shares (of the nonce-derivation key or of the nonce) in $\mathbb{F}_n$, and the bits (i.e., in $\mathbb{F}_2$) used in the distributed hash computation, a modular conversion mechanism can also be used. In some cases it can be easier to use a Q2 access structure, to handle multiplicative shares [BST21].

3. **Challenge:** compute the challenge χ as prescribed (see Table 5).

4. **Signature shares:** locally compute the signature share $S_i = r_i + \chi \cdot s_i \pmod n$ and send it to a *combiner* (anyone receiving all signature shares), who can then trivially obtain the final signature.

The main challenge above is the distributed SHA-based hashing needed to obtain the secret-shared nonce r, depending on the secret shares v_i of the nonce-derivation key v. The generic feasibility of MPC guarantees this is possible (e.g., see [BST21] for an implementation in an honest majority setting), albeit contrived when compared with what is needed for probabilistic Schnorr.

As an alternative, substantial efficiency improvements can be obtained by using an MPC-friendly hash (not the case of SHA-512 or SHAKE256) to distributively compute the nonce. This will no longer yield a *functionally equivalent* signature, but it will still be *interchangeable* w.r.t. EdDSA verification. Note that the hashing used to generate the challenge χ remains the original (SHA-based) one [BST21, §6].

4.3.3. Threshold signing with local deterministic contributions

An alternative solution to the key recovery pitfall (§4.3.1) is to have parties generate their nonce contributions deterministically and supply an accompanying proof that they were generated correctly [GKMN21; NRSW20].

1. **KeyGen:** Either a dealer or dealerless keygen protocol provides to each party a secret share s_i of the signing key. Each party i can locally select, independently, a nonce-derivation key v_i and send a commitment of it to all other parties. The parties may also generate some additional random state to be used for the proof of correct nonce derivation during the signing process. [GKMN21]

2. **Nonce commit:** Each party locally derives their deterministic contribution r_i for the

nonce, which depends on the secret v_i and on the public message M. Then the party commits as usual by sending $R_i = r_i \cdot G$ to everyone, but now also sends a ZKP that this is correctly related to the commitment of the nonce-derivation key. If all the proofs are valid, the honest parties combine the various contributions to obtain the global nonce commitment $R = \sum R_i$; otherwise, the parties abort. The specifics of the ZKP and deterministic function depend on the scheme.

3. **Challenge:** The challenge χ is computed as usual (see Table 5).

4. **Signature shares:** Generate, broadcast, and aggregate (partial) Schnorr signature shares. The actual techniques may be more sophisticated, such as by masking the typical signature share such that the masks are cancelled out when combined across parties [GKMN21], or including multiplicative coefficients that allows for key aggregation (in case of multi-signature) [NRSW20].

What distinguishes schemes with local deterministic nonces from each other is the pseudorandom function (PRF) used to generate the nonce, and the ZKP method for proving it was properly generated. In MuSig-DN [NRSW20] (a multi-signature scheme), the nonce is a specially designed PRF. It is keyed with the nonce derivation key, and takes as input the message M, the set of signers' public keys Q_i, and the commitments of the nonce derivation keys. The corresponding ZKP is computationally heavy, but signing takes only two rounds and is very efficient bandwidth wise. In [GKMN21], the PRF is the NIST-standardized **a**dvanced **e**ncryption **s**tandard (AES) cipher, and the ZKP is based on garbled circuits. This is computationally lighter, at the expense of higher bandwidth and three rounds of communication.

4.4. Probabilistic threshold Schnorr

The probabilistic approach for threshold Schnorr/EdDSA signing allows the distributed nonce generation to take advantage of homomorphic properties innate to the signature scheme elements. As mentioned (§4.2.1), the secret-sharing of a random secret nonce can be performed by a DKG protocol, then to be followed by a simple local generation of signature shares [SS01]. Some schemes can be tailored for a small number of parties, e.g., two [NKDM03]. More recent works have focused on a reduced number of communication rounds (though still making use of a broadcast channel, whose real implementation may require multiple rounds, depending on the system model). The protocol design can be framed within a simulatable (§4.4.1) or a game-based (§4.4.2) security formulation.

4.4.1. Simulatable threshold Schnorr in three rounds

In the ideal/real simulation paradigm of MPC, which allows for composability of ideal components, a threshold Schnorr protocol is relatively straightforward when considering as available gadgets an ideal commitment scheme, an ideal non-interactive zero-knowledge proof of knowledge (NIZKPoK), and assuming authenticated communication [Lin22]. The protocol follows from the intuitive semi-honest threshold Schnorr. A coordinator can be employed to decide the message to be signed and the signatory-subset ($\mathscr{P}'$), who collaboratively determine a session id (*sid*) for each signing execution. The signature format uses as first component the challenge χ, instead of the nonce commitment R, which technically makes the scheme not interchangeable w.r.t. EdDSA. However, the scheme could be adapted to become interchangeable.

1. **Keygen:** Based on a PKI, the parties are either given shares of the secret signing key or perform a Feldman VSS.
2. **Nonce commit:** Each party is invoked with the same message M to be signed.
   - **Agree on session identifier (*sid*).** Initially, each party P_i commits (in a hiding manner) to a share R_i of the usual Schnorr nonce commitment R [notice the double "commit"], at the same time that it proposes a contribution sid_i to a session id. Essentially, the double commitment prevents the nonce commitment itself from being biased/manipulated even by the last party to propose their contribution. The signatory-subset S is either assumed known or proposed by the central coordinator once hearing from the several parties. Each party calculates the session id *sid* for the ongoing signature protocol based on the signatory subset.
   - **Reveal the nonce commitment contributions R_i.** The nonce commitment contribution $R_i = r_i \cdot G$ (not the actual nonce contribution r_i) is then opened (verifiable w.r.t. its corresponding commitment) to the coordinator, along with a ZKPoK of the secret nonce contribution r_i, and a signature bound to the *sid*. The parties then homomorphically build the global nonce commitment R, by simple group sum of all corresponding shares.
3. **Challenge:** χ is computed as usual, based on R, Q and M (see Table 5).
4. **Signature shares:** The signature shares s_i are generated locally by each party, based on the calculated challenge, the signing key-share and the nonce-share r_i. The central coordinator (or anyone with access to the signature shares) can build the final signature and check its correctness.

The proof, in the static corruption model, relies on the simulation of ideal components, which allows extracting the hidden elements (e.g., nonce shares and signing-key shares) that enable ensuring the ideal execution is indistinguishable from a real one.

4.4.2. Probabilistic Two-Round Schnorr

A class of two-round threshold probabilistic Schnorr schemes [KG21; NRS21; AB21; CKM21] protects against the k-sum attack [DEFKLNS19] by using multiple nonce contributions per participant, and employing a "nonce binding" technique where each share of the nonce becomes dependent on the message, the set of cosigners, and the nonce contributions of all the cosigners.

These two-round protocols can precompute the first round, deferring for later a single round of communication for signing. The description below corresponds to FROST without preprocessing [KG21]. Other schemes operate in a similar manner.

1. **KeyGen:** Each party receives their signing key share s_i either from a dealer or via distributed key generation.
2. **Nonce commit:** The most distinctive aspect of this class of protocols is that each party generates two or more nonce contributions $(r_{i,1}, r_{i,2})$, instead of just one (usually two, but possibly more, depending on the scheme and security model). The contribution of each party to the final nonce commitment R is "bound" to the message M, the set $\mathscr{P}'$ of cosigners, and each of their own nonce commitments R_i supplied during a given signing operation.
Specifically, party P_i chooses two random nonces $(r_{i,1}, r_{i,2})$, and generates their corresponding commitments $(R_{i,1} = r_{i,1} \bullet G, R_{i,2} = r_{i,2} \bullet G)$. Let B be an ordered list of the participants involved in the signing operation, $\mathscr{P}'$ and their commitments: $B = \{(i, R_{i,1}, R_{i,2}) : i \in \mathscr{P}'\}$. All parties compute a set of "binding values" $\rho_i = H(i, M, B)$, for $i \in \mathscr{P}'$. The final nonce commitment R, common to all parties, is then $R = \sum_i R_{i,1} \boldsymbol{+} \rho_i \bullet R_{i,2}$. Given the linearity of the secret-sharing scheme, the corresponding implied secret share of the nonce for each party P_i is $r_i = r_{i,1} + \rho_i \cdot r_{i,2}$.
3. **Challenge:** χ is computed as usual, based on R, Q and M (see Table 5).
4. **Signature shares:** Each party's signature share S_i is computed as $S_i = r_{i,1} + \rho_i \cdot r_{i,2} + \chi \cdot \lambda_i \cdot s_i$, where λ_i is the Lagrange coefficient for the i-th cosigner in $\mathscr{P}'$.

The elaborate nonce commitment procedure is needed in order to thwart the k-sum attack [DEFKLNS19], which some two-round Schnorr multisignature schemes were susceptible to when an adversary could open multiple concurrent signing sessions. The attack involves finding a challenge value $\chi^* = H(R^*, Q^*, M^*)$ that is the sum of several other challenge values that differ in either the group's nonce commitment R or the message M. The attack is possible when the adversary has control over the nonce commitment R, by choosing the contribution of a corrupted party adaptively after seeing the contributions of all other parties. Exploiting the attack involves solving the Generalized Birthday Problem, which can be done with subexponential complexity using Wagner's algorithm [Wag02].

To turn the above scheme into a single round signing protocol, parties can locally generate a list of their nonce contributions and corresponding commitments, securely save them, and publish a list of commitments to a common location (or provide them to a party acting as the coordinator or signature aggregator). When a new signing session is initiated, the next set of commitments for each party can be sent to the parties along with the message.

The FROST scheme [KG21] is full threshold, meaning it can be instantiated with any secret-sharing recovery threshold t (out of n). MuSig2 [NRS21] and the delinearized witness multi-signatures (DWMS) [AB21] are multisignature schemes that operate in the plain public key model. SpeedyMuSig is similar to MuSig2 but operates in the KOSK model, which enables faster key aggregation [CKM21].

5. Further considerations

Section 4 has described various approaches for producing threshold Schnorr-style signatures. The present section proposes complementary aspects relevant for when preparing future related guidance and recommendations. These considerations are also relevant for any upcoming call for contributions and/or when analysing corresponding proposals of threshold schemes interchangeable w.r.t. EdDSA verification.

- Section 5.1 enumerates aspects of the threshold setting that make the case of a corrupted signer more complex and inherently more pertinent.
- Section 5.2 points out the diversity of security formulations, and how some security notions (e.g., strong unforgeability) are generalized in the threshold setting.
- Section 5.3 considers characteristics of the system model, namely assumptions about underlying communication functionalities.
- Section 5.4 revisits the issue of bad randomness, and how the threshold setting enabled new ways of resolving it.
- Section 5.5 motivates modularity and composability, and recalls useful phases (e.g., key-resharing, and replacement of faulty-parties).

5.1. "Thresholdized" signer

In the conventional setting, the security formulation of digital signatures is classically established by an *unforgeability* game (Section 3.3). There, the adversary does not know the private signing key but controls a client, who can request signatures from a signing oracle that knows the private key. The case of a corrupted signer is typically less considered, although it is the basis for the *message binding* property (Section 3.4). In the threshold setting, the signer becomes distributed, due to the secret sharing of the private key across multiple parties. The adversary can then also control some of the key-share holders. It thus becomes relevant to consider the case of a corrupted signer (i.e., in the threshold sense). The more complex adversarial model raises new considerations about adversarial capabilities and goals. For example:

1. **Corruption threshold:** the adversary can control up to a corruption threshold f of the key-share holders. Which ranges of f are acceptable? Some functionalities/protocols will only work for certain intervals of the proportion f/n (corruption threshold over number of parties).
2. **Agreement:** the decision to sign a message becomes distributed across a set of cosigners, including corrupted parties. Whether the agreement is assumed as implicit, or follows explicitly from a verifiable request from an external client or coordinator, it needs to actually be implemented when the system is deployed.
3. **Number of signatures:** if the participation threshold (i.e., the needed quorum) is not higher than $(n+f)/2$, how many signatures should it be possible to create from a single authorized request that is broadcast to all parties? Consider an adversary who — besides compromising f parties — has some control over the network, and can

partition the honest parties into two separate networks, causing them to participate in two distinct Schnorr signings of the same message.

4. **Concurrent signing:** the adversary can corrupt some of the parties and thus observe and interfere with the intermediate steps of the concurrent generation of multiple signatures. This is not possible in the conventional unforgeability game, where each signature is produced by an oracle who processes each request independently. Without proper safeguards, some protocols secure in a threshold *standalone* setting (without concurrency) may enable forgeries in the threshold concurrent setting. For example, if the signature scheme allows the adversary to maliciously influence the nonce commitment R, then a forgery may be obtained upon solving a k-sum problem [DEFKLNS19; BLLOR21].
5. **Messages adapted to the nonce commitment:** depending on the threshold protocol, an adversary may be able to select a message with a noticeable relation to M (e.g., M=R). This would not be possible in the conventional SUF game, since there the oracle signer produces a (pseudo)random R. However, actual unforgeability may follow even though the adversary is able to learn R before selecting the message M. (Note that such capability is not considered in usual conventional proofs of security.) Other security formulations may specifically disallow this.

Being aware of the possible options and their differences is relevant to enable a security formulation that captures the intended functionality and/or desired properties.

5.2. Threshold security formulation

There is room for nuances in security formulations for threshold signatures. For example, an ideal threshold signature functionality — in the **u**niversal **c**omposability (UC) framework — may define that a signature is produced only when all parties request the signature of a given message M. In a security-with-abort formulation, the adversary is allowed to see the signature first and decide whether or not the honest parties can receive it [BST21, Fig. 8].

The functionality may also require that all parties agree on a proposed nonce commitment R, before proceeding to release the remainder (S) of the signature [GKMN21, Func. 9.1]. The quorum (participation threshold) t' and session identifier *sid* may be explicitly encoded, so that the signature is produced once t' parties request it, with an agreeing *sid* [Lin22, Fig. 4.2].

Security formulations can be described via an ideal functionality or via games for each intended property. These may also encode whether or not, for example, a coordinator/aggregator facilitates the communication between the remaining parties, and is responsible for outputting the final signature upon obtaining signature shares from the other parties [KG21; Lin22]. In the case of multi-signatures, the UF game also considers the set of public keys used to generate a signature. Then, an adversarial win requires generating a signature for a message M and a cosigners set $\mathscr{P}'$ (i.e., set of their public keys) that includes at least one honest party that never agreed to sign the message within that cosigners set ([BN06, Sec. 4]; [NRS21, Fig. 3]). This can be generalized to a SUF sense, by considering as forgery any new signature for the same pair ($\mathscr{P}'$, M). Various levels of unforgeability strength can be defined based on the goal and capabilities of the adversary (see Sections 3.3 and 5.2.1), namely what is considered a valid forgery and which contributions an adversary can obtain from honest parties [BTZ22; BCKMTZ22]. Security formulations can also cover additional modules/features, such as robustness [RRJSS22].

The suitability of each formulation can vary with the intended system model and/or the presence of features of interest to envisioned application settings. It is nonetheless important to check whether the adversary in the threshold setting is prevented from gaining an ability that exceeds that of the adversary in the conventional scheme.

The simulatability setting provides a natural way of going beyond unforgeability. For example, it inherently requires an unbiased nonce commitment, whereas in the case of a game-based definition for UF, that property only tends to appear as a protection against a concrete attack. Still, one can also define games for other threshold properties. As another example, when an ideal functionality directly selects the nonce commitment, after the message to be signed has been determined, the formulation inherently requires protects against subliminal channels (see §5.2.4).

5.2.1. Strong threshold unforgeability

Since EdDSA is not verifiably deterministic, unforgeability should be considered in the "strong" sense: SUF (see Section 3.3). This notion becomes generalized in the threshold setting, where the adversary can corrupt up to f parties, besides possibly controlling a client able to issue valid requests for message signing, and also possibly controlling the message delivery in some channels. Thus, with EdDSA being SUF in the conventional setting, it is useful that a threshold scheme interchangeable w.r.t. EdDSA considers a threshold notion of SUF within the claimed corruption threshold.

Recent work has formalized game-based definitions for various levels of strong unforgeability in the threshold setting [BTZ22; BCKMTZ22]. The different levels consider, for example, the number of honest parties providing contributions (e.g., signature shares, if in a non-interactive setting) upon receiving a signing request. Also of interest are simulatability formulations, where an intended notion of unforgeability (as well as other properties) may be derived from the specification of an ideal functionality.

A SUF notion should clarify the conditions under which an adversary is expected to be able to generate a new signature (see §5.2.2). Also, unforgeability should remain even when the adversary is able to adaptively corrupt parties (see §5.2.3).

5.2.2. Number of signatures per request

The conventional unforgeability notion asks that an adversary be unable to obtain more signatures than those that have been properly "requested". In the threshold setting, the notion of "request" can depend on the system model. For example, it can vary between (i) being any request signed by an authorized client (including one controlled by the adversary), and (ii) being the result of an agreement (i.e, decided by an external protocol) between the parties. There are diverse options for the threshold security formulation to encode the meaning of valid signing request.

A security formulation for a threshold signature scheme should enable a clear understanding of what would be considered generating too many signatures, as compared to the number of legitimate requests. It should consider that some malicious requests (say, in the model where a client directly sends valid signed requests to each separate party) may lead to partial executions that do not end with a valid signature. These partially fulfilled requests (a notion not present in the conventional setting) should not give the adversary an additional *advantage* in producing non-requested signatures. Several techniques may be considered to protect against the generation of extra signatures. These may include, for example, a requirement for starting with a collective agreement on which messages to sign, and in which order, and/or the use of clocks, timestamps, counters and session identifiers.

The notion of a *participation threshold* t' is also relevant. Consider a protocol with a small *corruption threshold* f (with $f < \lfloor n/2 \rfloor$), and an underlying secret-sharing whose *reconstruction* threshold t is equal to just one more party (i.e., $t = f + 1$). If a request does not identify the cosigner subset $\mathscr{P}'$, then an adversary controlling the network channels can partition the set of parties into two independent quorums. Could this lead the same request to generate two different signatures? Despite the low *corruption* threshold f, by requiring that the *participation* threshold t' is higher than $n/2 + f$ (the exact minimum may vary with the type of synchrony and other assumptions), then any two signing executions will have at least one common honest party. Note that this is exemplifying a *participation* threshold t' higher than the *reconstruction* threshold t. Alternatively, a protocol may require that each signing request explicitly identifies the subset $\mathscr{P}'$ of allowed cosigners, to prevent non-included honest parties from giving a contribution to the adversary.

Example of multiple uncontextualized requests. Consider an application that composes a threshold signature scheme with an external decision algorithm used by each honest party to decide whether (i) to participate honestly in the signing, or (ii) to declare not being available to participate. What happens then if a request to sign the same message appears several times, while the parties' participation decisions (whether or not to sign that message) alternate across requests and across parties?

Consider a threshold scheme with participation threshold $t = 3$, and only $n = 3$ parties: A and B are honest; C is malicious. Suppose there are two certified requests to sign the same message M. Suppose that upon the first request only parties A and C are willing to participate, and upon the second request only parties B and C are willing to participate. Suppose the adversary is able to replay messages, judiciously selecting which messages to send to which honest parties. Can the adversary induce the creation of a signature, even though the number of "honest" parties available to participate for each request has never reached the participation threshold?

A proposed security formulation and system model for a threshold signature scheme should include details that enable answering this type of question. Special care is required for the case of concurrent signing requests, where parties may receive requests in inconsistent orderings, or even with inconsistent content produced by malicious participants. The use of (and agreement on) session identifiers is often a necessary element to handle concurrency.

5.2.3. Safety against adaptive corruptions

Unforgeability should be guaranteed against adversaries that can *adaptively* choose, based on an observation of the protocol execution, which parties to corrupt (up to the threshold f). As compared to static corruptions, which must occur at the onset of a protocol execution, adaptive corruptions introduce a new degree of freedom.

The following is a classical example (slightly adapted in the parameters) of a statically-secure but adaptively-insecure protocol [CFGN96], w.r.t. confidentiality for secret storage: an incorruptible dealer distributes secret-shares of a key across a relatively small random subset with f parties ($f \approx \sqrt{n}$), using an f-out-of-f secret-sharing scheme, and then advertises the subset. A static adversary has a negligible probability — asymptotically in n — of having corrupted the needed subset of f parties before said set is advertised. Concretely, the probability is the inverse of the number ("n choose f") of possible subsets of f parties from within the set of n parties. Conversely, an adaptive adversary can wait to hear the advertisement and only then corrupt exactly the f key-share holders, thereby finding the secret. The example can be adapted to other safety properties, such as unforgeability.

Despite the gap between static and adaptive security, many protocols that in practice are proven statically-secure also retain some desirable properties (though not necessarily all) in the adaptive corruption setting. W.r.t. a game-based security property, a proof of security for a given protocol may happen to be independent of the difference between static vs. adaptive corruptions, and imply security against both types of adversaries. In the UC simulatability setting (ideal/real simulation paradigm) [Can01], security against adaptive-active corruptions is in general more challenging to achieve, compared to the case of static-active corruptions [CDDIM01]. However, this difficulty is often because the security formulation comprises not just one safety property (such as unforgeability), but rather defines a whole functionality encompassing properties of a different nature, such as deniability of execution and composability (which are not captured by the unforgeability game).

Because of the technical difficulties with adaptive security in a simulatability setting in the UC framework, it is common to see protocols proven secure only in the static setting, often with an implicit understanding that the lack of adaptive security does not mean a complete breakdown of safety properties in case of adaptive corruptions. In fact, a loss of deniability of execution and/or of some types of composability is something that may already happen when a protocol deployed in practice uses real (non-ideal) components to instantiate ideal components used in the proof of security (e.g., replacing a programmable random oracle by a cryptographic hash function).

Given the possibility of adaptive corruptions in the real world, it is important to consider for any proposed threshold signature scheme whether the major safety properties of interest (such as unforgeability) are safeguarded against such an adversary. It is acceptable that this comes at the expense of some adjustment of the ideal functionality. It can also come in the form of a different argument, such as the case of adaptive security in the constructive cryptography (CC) setting [HLM21]. The latter provides an approach to explore another flavor of simulatable adaptive security, while avoiding the mentioned difficulties.

5.2.4. Preventing subliminal exfiltration

A (stateless) probabilistic signature scheme provides an avenue for exfiltration of secret information, using the randomized component (e.g., a nonce commitment) as a subliminal channel [Sim94; AMV15]. This also applies to deterministic signatures (such as EdDSA and deterministic ECDSA) that can be undetectably made probabilistic by a malicious signer. For example, consider the case of a client who requests EdDSA signatures from a cryptographic security module (CSM) that holds the signing key. If the CSM has been corrupted, then it can maliciously influence the nonce commitment R to exfiltrate secrets via signatures.

The threat of subliminal channels can be mitigated with suitable threshold schemes, for both probabilistic and deterministic schemes. Suppose the administrator establishes a threshold signature scheme across various CSMs. A protocol can be such that no isolated CSM (nor any coalition up to the corruption threshold) is able to bias the bits in the final signature. A limitation exists in the case of security-with-abort formulations, where the the adversary has a chance to prevent undesired outputs, which provides a low capacity channel.

Since unforgeability does not imply unbiased signatures, the threshold assurance of the latter in Schnorr/EdDSA signatures depends on the actual threshold scheme / security formulation. In particular, the malicious manipulation is allowed in some 2-round protocols (§4.4.2) where a malicious party (possibly the coordinator) is able to wait to be the last to propose a nonce commitment contribution, while already knowing the nonce commitment contributions of the other parties. Conversely, threshold schemes arising from a simulatability formulation tend to automatically ensure an unbiased nonce commitment. This is because their ideal functionality, which the protocol needs to emulate, selects the nonce r (uniformly or pseudorandomly) and calculates the nonce commitment R without interference from any party. This applies to both probabilistic (§4.4.1) and deterministic cases (§4.3). Naturally, this is also possible from protocols proven unforgeable with respect to a game-based definition, such as usual in those with three or more rounds [SS01; MPSW19].

5.3. System model

Several elements of the system model affect the suitability of protocols, approaches and realizable functionalities for threshold signatures. The following are relevant considerations: how *authenticated channels* are implemented (§5.3.2); whether parties have access to a *reliable broadcast* channel (§5.3.3); which *timing assumptions* the protocol can rely on (§5.3.4);

whether the deployment allows for a *precomputation* (offline) phase, before learning the message to sign (§5.3.5); what happens when system model assumptions are broken (§5.3.6).

5.3.1. Interface for signature request and delivery

Use of a coordinator or aggregator. Important safety properties, such as unforgeability, must hold even if the coordinator is malicious and sends inconsistent messages across different cosigners, so long as the number of corrupted cosigners is within the corruption threshold. As a tradeoff, some availability properties may be sacrificed, as happens with security-with-abort formulations, where an adversary can decide to not let the protocol produce a valid signature.

Shared-I/O modes. In a threshold scheme where the operation request comes from an external party, it is possible to have none of the internal parties (i.e., the key-share holders, including corrupted ones) see the final signature value. This shared-output (shared-O) mode can result from a formulation where the ideal functionality sends the signature (or set of signature shares) only to the client that requested it. The ideal functionality also interacts with the various parties to ask their agreement about signing the message. However, besides seeing the message, each party sees at most a few shares of the signature (not enough to reconstruct it). A shared-Input (shared-I) mode is also conceivable, with the input arriving secret-shared (e.g., possibly with a VSS to enable verifying consistency of the shares). This is less practical since it requires a distributed computation of the SHA-based challenge χ. The shared-I/O modes [NISTIR 8214A, §2.3] are not meant to include the special case of an MPC where the message remains secret for the entire threshold entity (even if all parties collude). The current scope is to consider an outsourced signature, performed by a threshold entity, where the client (the signature requester) may at most perform secret-sharing and/or reconstruction. Naturally, one can combine both shared-O and shared-I features into a shared-IO mode.

Threshold auditability. Threshold schemes may have additional features beyond their functional output. For example, public auditability may be useful for some applications. This verifiability can be embedded into secret sharing [Sch99] as well as into more general MPC [BDO14]. Besides the original intended output, a publicly auditable MPC would produce a proof of correct execution. For a threshold signature scheme this could mean a proof that a signature was produced via a threshold interaction ([NISTIR 8214A, §2.5]), with the agreement and collaboration of a particular subset of parties. This makes sense if the client or the public has access to a PKI with the public keys of the cosigners, or to something that verifies the underlying secret sharing. To be clear, an auditability transcript would not be considered part of the signature to be parsed by the client, but rather an auxiliary output of the protocol execution, to possibly be consumed by a separate audit application.

5.3.2. Authenticated channels

It is customary in MPC protocols to assume the existence of authenticated channels. In a practical deployment, the channels have to somehow be instantiated. In the real world, authentication may depend on physical assumptions (such as a communication wire connecting two parties) and/or cryptography. Such a setup may be prepared by an administrator (e.g., if all parties belong to the same administrative domain), or in some ad-hoc manner (perhaps based on a PKI, a distributed protocol, or other means). Practical implementations can, for example, be based on:

- public-key cryptography: e.g., based on digital signatures with one public-key associated to each party; or
- symmetric-key cryptography: e.g., using a **h**ash-based **m**essage **a**uthentication **c**ode (HMAC), with a different key for each pair of parties.

The type of authentication affects the security and capabilities of protocols. For example, a PKI can support transferable authentication based on signatures, so that party A can prove to party B that party C sent something to A. Conversely, an HMAC-based authentication is typically non-transferable (deniable). Typical ideal authenticated channels are deniable. A transferable instantiation may be considered a feature or a handicap, depending on the context. Authentication is also relevant between a client that requests a message and the parties that receive such request.

The actual (real) authenticated channels available in the signing phase may be different from those in a preceding distributed keygen. In fact, the state obtained from a keygen (distributed or dealer-based) may be designed to enable new authenticated channels (even private, if need be, to support secrecy of transmitted content) in the subsequent signing phase. This requires proper care, or else possibly result in a security failure. In practice, a popular instantiation of authenticated and/or private channels is based on the **t**ransport **l**ayer **s**ecurity (TLS) protocol. However, its composability with (i.e., replacing the ideal authenticated channels of) a threshold scheme should be carefully considered. For example, a careless instantiation of authenticated channels by using the actual key-shares obtained in the keygen phase could help the adversary produce a forgery. Conversely, it is an interesting consideration to think how to enable, in the signing phase of a threshold signature scheme, an instantiation of authenticated channels based on the material obtained during the keygen phase. Conceivably, this may be based on signatures that rely on the actual key-shares of the signing key, or derived therefrom.

As part of the communication setup, a threshold scheme specification can assume that every party knows the set of possible cosigners (and each other's public key, or pairwise symmetric-key). In practice this may be bootstrapped by an administrator, or by an ad-hoc agreement between parties with the help of a PKI. Some system models may allow a dynamic set of participants, establishing rules for deciding when and how to onboard new cosigners (and their keys), and/or remove old cosigners.

5.3.3. Broadcast

As a primitive to facilitate obtaining agreement, some protocols make use of *reliable broadcast*, where an honest receiving party is ensured that other honest parties have also received the same message. In some cases, reliable broadcast may be woven into the communication steps of the signing protocol, to reduce the overall number of communication rounds. Its realization depends on the communication model, e.g., whether or not there is a PKI to enable transferable authentication of messages (i.e., party A can prove to party B that party C has signed a message), a coordinator facilitating the message delivery (possibly also signing the delivered messages) vs. only point-to-point channels. The notion of reliable broadcast is stronger than a simple multicast where a party sends a message to every other party. In other words, it matters that each receiving party gains assurance that a particular message claimed to have been broadcast/multicast has also been received by every other honest party.

5.3.4. Timing assumptions

The performance and security of threshold signature schemes depends on the underlying communication model, particularly the timing for message delivery across participants. In the synchronous model, there is a known upper bound on the delay before messages are delivered. The asynchronous model, on the other hand, has no upper bound on the message delay, only requiring that messages be delivered eventually. A variety of other models exist, such as partial synchrony, where a period of asynchrony is followed by a period of synchronous communication.

More conservative timing assumptions can make a protocol more resilient to problems with the underlying communication network. However, this can come at the cost of stricter requirements on the protocol's design, performance penalties, and lower corruption thresholds. For example, the asynchronous setting does not allow parties to distinguish between the following scenarios: (i) a malicious party did not send a message, and (ii) an honest party sent a message, but is experiencing delays in its delivery over the network. As a result, more honest parties may be required to achieve the protocol's security goals.

5.3.5. Offline/online phases

Efficiency goals usually aim for low latency (low round-complexity), low communication complexity (number of communicated bytes) and/or high throughput (number of signatures per unit of time). In threshold settings, there are so-called offline/online models that allow pre-processing a significant amount of computation and communication in an *offline* phase, before the actual arrival of a message signing request. This allows for a subsequent lighter/faster *online* phase. For example, the selection of elements necessary for a later determination of the nonce commitment R and the nonce secret-sharing $[r]$ can be performed before the message is known. (Note that even the contributions R_i to the "nonce commitment" R may be initially "committed", when a security formulation requires preventing the adversary from maliciously affecting R.) The generation of correlated randomness (and pseudorandomness) can be particularly useful [Bea96; IKMOP13; BCGIKS19]. An offline phase may also prepare some aspects of agreement, such as possibly a coordinator.

5.3.6. Beyond covered assumptions

A threshold scheme may be designed and have provable security for a particular system model and adversarial capabilities. What happens, however, if those assumptions are not met? For example, what happens if (i) an assumed synchronous communication network turns out to be asynchronous (see §5.3.4), or if (ii) an assumed reliable broadcast channel (see §5.3.3) does not actually reach every party, or if (iii) the number of corrupted parties (see Section 5.1) exceeds the corruption threshold by 1 or more? It is useful that the security analysis of a threshold scheme considers these questions, identifying possible ranges of graceful degradation, vs. others of complete security breakdown. In the case of a signature scheme, allowing forgeries would be a complete security breakdown, whereas losing fairness could be acceptable. Thus, if a protocol enables a given security formulation with up to f corruptions, it may still enable another security formulation with up to $f+\varepsilon$ corruptions, possibly with mixed types of corruptions (some active, others fail-stop, others semi-honest) [FHM98; HM20; DER21]. Graceful degradation w.r.t. to continued corruptions can also be promoted by abort-recovery subprotocols, for example if identifying the parties that have misbehaved and then being able to remove them.

5.4. Good vs. bad randomness

The issue of good vs. bad randomness is central to implementation security, as already discussed in Section 3.5. In the threshold setting, each party may be subject to the causes of bad randomness that affect the conventional (non-threshold) setting, such as insufficient entropy, or rewinding/snapshot susceptibility (see §3.5.1). In the conventional setting, these concerns have motivated the use of pseudorandomness when generating the secret nonce in EdDSA. The use of randomness is more complex in the threshold setting, with both more opportunities and challenges for security.

A naive recourse to a purely pseudorandom mode may be vulnerable to the malicious introduction of randomness (see §4.3.1). Conversely, the threshold setting can provide some protection against bad randomness in probabilistic signature schemes. For example, a threshold protocol can combine various random contributions in such a way that the good randomness from a single honest party results in a signature without bias. Probabilistic threshold signature schemes nevertheless have various randomness-related concerns, such as: inadvertent correlated randomness across parties (§5.4.1), attempts to maliciously influence the value of the secret nonce r or its commitment R (§5.4.2), and internal attacks against internal "well behaved" parties that have bad randomness (§5.4.3).

Issues of bad randomness can affect even threshold protocols for deterministic signing. This is because multi-party protocols often resort to randomness for internal gadgets (e.g., garbled circuits and oblivious transfer). In fact, even secret sharing of a key most often relies on randomness. Therefore, the issue of good vs. bad randomness needs to be carefully considered in the specification of a threshold scheme, including the phases of keygen and signing (both deterministic and probabilistic).

5.4.1. Inadvertent correlated randomness

The threshold setting brings in the issue of inadvertent "correlated randomness". When various signers operate in a similar environment (e.g., same software bootstrapped in equal conditions, and/or using a common pool of entropy), their resulting local randomness may be inadvertently correlated.

One mitigation to address this unwanted correlation is to have each party transform their randomness by applying a pseudorandom transformation relying on a local secret. This ensures that the randomness of each party is unpredictable, as long as their secret remains unpredictable to the other parties. For better resistance against side-channel attacks that may try to exfiltrate such a secret, the secret can be updated in each use (to the extent that that party is able to maintain that extra state).

The issue of inadvertent "correlated randomness" discussed here should not be confused with the use of securely generated "correlated randomness" in MPC [IKMOP13], which can be useful to reduce communication complexity.

5.4.2. Manipulating the nonce commitment

It is well known that the biasing of the secret nonce r used to produce an EdDSA signature allows extracting the signing key (§3.5.2). More subtly, the possibility of malicious influence of the nonce commitment R is also problematic. If a cosigner is able to present their contribution R_i once already able to compute the final nonce commitment R, then it can use the nonce commitment as a subliminal channel to exfiltrate information. Perhaps more importantly, in the threshold setting, the manipulation of the nonce commitment can in some cases enable forgeries, in the case of concurrent signing [DEFKLNS19]. The malicious influence can be avoided by requiring that every participant commits to their contribution before anyone reveals it [SS01; MPSW19] (see also §5.2.4). These challenges should be limited within the indicated corruption threshold (see Section 5.2.4), since the R is supposed to be indistinguishable from random.

5.4.3. "Well-behaved" parties with bad randomness

The threshold setting can easily leverage the local good randomness from a single participant to ensure an unbiased secret nonce r, and thus mitigate the risk of leaking information about the signing key. The tolerance to malicious corruptions already handles the case of (up to a threshold f) parties with bad randomness. Yet, there is benefit in focusing attention in the specific case of "**w**ell-**b**ehaved but with **b**ad local **r**andomness" (WBBR) parties.

Corruption escalation. The key-recovery pitfall described earlier (§4.3.1), for a (careless) threshold deterministic scheme can be reconsidered for a probabilistic scheme. The former had an honest deterministic party interacting with a maliciously randomized party. By analogy, the same issue may occur in a (careless) probabilistic scheme where a well-behaved (colloquially called "honest") party only has access to "bad randomness" (such as from a repeating seed). Then, the well-behaved party may leak their secret key-share to an internal malicious party, becoming itself corrupted to a higher degree. A threshold protocol should protect WBBR parties from having their corruption escalate to an exfiltration of their key-shares when interacting with (other) malicious parties.

Tolerance to more corruptions. If handled properly, the attention to the WBBR case allows for a possible increase in the tolerance to corruptions. Once fixing the main corruption threshold f for malicious compromises, the requirement for "good randomness" may be sufficient to apply to a threshold of the remaining honest parties, rather than to all of them. This can mean more suitability for deployment in settings where some "bad randomness" is expected. The WBBR parties would then leverage good randomness from the other honest parties. The advantage is resistance to not only up to f (the original threshold of) arbitrarily malicious parties, but potentially to the participation of additional WBBR parties. The presence of at least one honest party (with good randomness) requires the number of WBBR parties to not be higher than $t' - f - 1$, where t' is the quorum required by for signing. In the optimistic case where every party follows the protocol specification, the good randomness from a single honest party, is sufficient to ensure an unbiased nonce, despite the possible presence of up to $t' - 1$ WBBR parties. Lower thresholds for WBBR may be required, depending on the approach, and a formal security claim requires careful analysis.

5.5. Modularity and composability

A threshold scheme proposal can benefit from a modular description and implementation. This applies both to protocol phases and to building blocks (gadgets).

5.5.1. Phases

Some modularity naturally follows from the structure of a signature scheme. The keygen and the signing phases should be defined separately, albeit in an interoperable manner. That is, the signing protocol should make sense regardless of whether the keygen is achieved via a dealer or a distributed protocol (§4.2.1).

Modularity also makes sense w.r.t. possible additional sub-protocols, such as:

- secret-resharing, for proactive security, to render useless any key-share that may have already leaked to the adversary (assuming fewer than f shares have leaked since the most recent resharing);
- dynamic change of participants, such as altering the set of potential cosigners (and thus, when applicable, their keys) and possibly the change of corruption and participation thresholds.

The above examples require deletion of old shares, in order to retain security in the face of mobile adversaries that continue corrupting parties after a resharing phase.

Some phases may be the result of a certified administrative request built in to the implementation, such as to increase n and f, requiring a resharing of the private key.Some phases may be activated when special internal conditions are met, such as those foreseen in threshold schemes with *identifiable abort*, where a party may be identified as malicious. A protocol may have a special provision to retry a signing operation after getting rid of an identified malicious party, in order to provide *robustness*, i.e., successfully producing a signature despite the malicious parties.

Each phase may come with tradeoffs, such as possibly imposing a more restrictive set of security parameters (e.g., thresholds) or setup conditions (e.g., communication network). For example, the phases may be more difficult to complete in an asynchronous network or against adaptive adversaries, and may bring other operational concerns related to agreement. Some changes in the system need to be agreed upon by all (or a qualified majority of the) honest parties, to avoid a partitioning where a qualified set of parties (able to produce signatures) retains a vision of the past shares.

Even within the signing phase, there may be a partition between precomputation and online sub-phases. There may also be a modular description of possible consensus mechanisms used to decide which message is to be signed, the session identifier *sid* and the subset of cosigners to participate in the session. Despite modular descriptions of some aspects of the signing phase, it may be possible to superpose them in order to reduce the number of rounds of communication. For example, the parties may both commit to their nonces and agree on an *sid* in the same round.

5.5.2. Gadgets

Ideally, various building blocks (gadgets) can be identified and used in a way that allows replacement with other instantiations, and/or which can be reused in other threshold schemes. This document has mentioned several examples of gadgets: secret sharing, garbled circuits, oblivious transfer, commitment schemes, secret resharing, Lagrange interpolation, zero-knowledge proofs, etc. The security upon replacement of a gadget instantiation by another one may depend on the composability of the scheme, as well as variations in the setup assumptions. Some replacements are safeguarded by some type of security proof (e.g., universal composability, where an ideal component can be replaced by a corresponding UC-secure one), while others may require a closer look (e.g., because of a somewhat distinct interface) but still provide a conceptual simplification that eases the analysis.

6. Conclusions

This document has discussed threshold signature schemes interchangeable w.r.t. the EdDSA verification specified in Draft FIPS 186-5. These threshold signatures allow for a drop-in replacement of conventional (non-threshold) EdDSA signatures, being compatible with legacy code for signature verification. Compared to conventional implementations, a threshold signature scheme enables a distribution of trust regarding the secrecy of the private signing key. The threshold setting additionally allows for better implementation security w.r.t. concerns of bad randomness and side-channel attacks (see Table 9).

Table 9. Types of signature vs. concern — informal assessment

Signature mode	**Nonce generation**	**Attack of Concern**	**Informal assessment**	
			Conventional	**Threshold**
Deterministic	Pseudorandom	Bias	Not applicable	Not applicable
		Side channel	More vulnerable	**Safer**
Probabilistic	Randomized	Bias	Vulnerable	**Safer**
		Side channel	Less vulnerable	**Safer**
	Hybrid	Bias	Not applicable	Not applicable
		Side channel	Less vulnerable	**Safer**

The use of "Less" and "More" preceding "vulnerable" is only for comparison within the side-channel attack concern. Each "Safer" is meant in comparison with the assessment of the conventional setting in the same row. In the threshold setting, the assessment does not relate to the corruptibility of individual parties, but rather to unforgeability property when assumed that the number of corrupted parties is within the allowed threshold. This informal table is meant only to provide intuition; more context is needed for formal conclusions about each concrete signature scheme.

6.1. Comparing probabilistic and deterministic threshold EdDSA

There is a wide design space for threshold signature schemes interchangeable w.r.t. FIPS-specified EdDSA verification. This includes schemes that produce deterministic signatures (though not verifiably-deterministic) and also probabilistic schemes. Considering the diversity of approaches and tradeoffs, it would be beneficial to devise recommendations or guidance, to facilitate the secure deployment of threshold signatures. This should involve a more thorough analysis and refined characterization of the potential space, aided by the broader community of cryptography experts.

Threshold deterministic EdDSA signatures may be useful in some niche cases, but they tend to be considerably less efficient than threshold probabilistic schemes. If an application requires ECC-based deterministic signatures interchangeable w.r.t. FIPS-specified verification, then the threshold setting provides an interesting mitigation against the lack of verifiable determinism. A protocol can be devised so that determinism stems from the coverage of the threshold corruption assumption, though this determinism remains unverifiable.

Deterministic threshold schemes that require a distributed SHA-based nonce computation are prone to an inefficient protocol. Other approaches that calculate a deterministic secret nonce using MPC/ZKP-friendly hashes can reduce the cost. In the setting of threshold signature schemes interchangeable w.r.t. EdDSA-verification, the probabilistic approach enables schemes that may be simpler and more efficient than deterministic ones. Intuitively, the probabilistic approach is natural for threshold Schnorr-style schemes, taking advantage of homomorphic properties already innate to the signature scheme elements.

Compared to probabilistic Schnorr/EdDSA schemes in the conventional setting, the threshold setting enables schemes that may be less vulnerable to biased random number generators. Additional assurance can come from utilizing a hybrid mode of nonce generation (see Section 3.5), which is possible in both conventional and threshold settings. It can be straightforwardly employed to enhance a prior use of pure randomness, by additionally applying a pseudorandom transformation, while retaining high efficiency.

The comparison between probabilistic and deterministic approaches can further depend on the application setting and intended features. For example, the resharing of a secret-shared private nonce-derivation key (only needed for the deterministic approach) may be substantially more difficult than that of the private signing key.

The mentioned features make probabilistic EdDSA well aligned for consideration by NIST, as framed in Draft FIPS 186-5 when expressing (page 5, end of item 3) that "additional digital signature schemes may be specified and approved in FIPS publications or in NIST Special Publications." Interestingly, Draft FIPS 186-5 already specifies probabilistic ECC-based signatures in the form of probabilistic E**C**DSA (which is more difficult to thresholdize). The consideration of probabilistic E**d**DSA for the threshold setting warrants a thorough analysis, as can take place based on a public call for threshold signature schemes interchangeable with EdDSA verification. The resulting analysis may clarify the potential and feasibility for adoption of threshold schemes for EdDSA.

6.2. State of the art and beyond

The state of the art in threshold schemes has come a long way, including progress in recent years with newly proposed schemes, and a better understanding of security (namely in the concurrent setting). At the same time, there remain worthwhile directions for future work. The following list summarizes possible features that could benefit from further attention from the community. While these are not necessary in order to have useful threshold signatures, they may have utility for some applications.

1. **Leveraging good randomness.** Schemes that leverage the good randomness from some participating honest parties, being secure even if other "well behaved" parties

(beyond the corruption threshold) have bad randomness (see §5.4).

2. **Authenticated channels with real keys.** A threshold scheme whose authenticated channels during the signing phase are based on signatures (possibly EdDSA/Schnorr) whose keys are determined in the (possibly augmented) keygen phase (see §5.3.2). Such a composition requires careful security analysis.

3. **Shared I/O.** Threshold signing where the parties do not get to learn the message being signed or/nor the produced signature (see §5.3.1).

4. **Adaptive simulatability.** An efficient/practical simulatable threshold scheme with proven strong unforgeability against adaptive corruptions, possibly in the constructive cryptography sense (see §5.2.3).

5. **Auditability.** Protocols that generate an auditable proof that the signature was indeed produced by a valid threshold interaction (see §5.3.1).

6.3. Recommendation for a public call for threshold EdDSA schemes

A public call for threshold signature schemes interchangeable with the standardized Draft FIPS 186-5 EdDSA verification could be of great benefit. It would seek to collect reference implementations, accompanied by technical explanation and security analysis. The scope would include threshold schemes for probabilistic signatures, as well as those with pseudorandom nonce generation. Such a call would need to provide baseline criteria [Call2021a], such as requiring a proof of active security with a minimum requirement of strong unforgeability. It should also be flexible to allow submissions across various ranges of number of parties and thresholds, security formulations (see Section 5.2), and system models (see Section 5.3). Ideally, the distributed computation would be based on cryptographic assumptions close to those required for EdDSA security, such as discrete-log and hash-related assumptions. Naturally, the interest on threshold schemes includes those for other NIST-approved key-based cryptographic primitives, including RSA, ECDSA and AES.

Besides the keygen and signing phases, it is useful to consider secret-resharing for proactive security, possibly also allowing *dynamic* change of the threshold parameters and number of parties. The envisioned call should recommend submissions to be described and implemented with modularity w.r.t. building blocks (gadgets) that are likely reusable by other schemes, or that can have different internal instantiations while having a similar interface. The security analysis should describe the security fall-back guarantees or breakdown when some of the operational requirements are not met (e.g., exceeded corruption threshold, asynchrony or non-reliable message transmission).

References

[**AB21**] Handan Klnç Alper and Jeffrey Burdges. "Two-round trip schnorr multi-signatures via delinearized witnesses". In: *Advances in Cryptology — CRYPTO 2021*. Springer. 2021. DOI: 10.1007/978-3-030-84242-0_7. Also at ia.cr/2020/1245 (Cited on pp. 29, 30).

[**ABFJLM18**] Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. "Differential attacks on deterministic signatures". In: *Cryptographers Track at the RSA Conference*. Springer. 2018. Also at ia.cr/2017/975 (Cited on p. 20).

[**AMV15**] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. "Subversion-Resilient Signature Schemes". In: *Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Association for Computing Machinery, 2015. DOI: 10.1145/2810103.2813635. Also at ia.cr/2015/517 (Cited on p. 36).

[**ANTTY20**] Diego F Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. "Ladderleak: Breaking ECDSA with Less Than One Bit of Nonce Leakage". In: *Proc. 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. ACM, 2020. DOI: doi/10.1145/3372297.3417268. Also at ia.cr/2020/615 (Cited on pp. 19, 20).

[**BCGIKS19**] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Efficient Pseudorandom Correlation Generators: Silent OT Extension and More". In: *Advances in Cryptology — CRYPTO 2019*. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-26954-8_16. Also at ia.cr/2019/448 (Cited on p. 40).

[**BCJZ21**] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. "The Provable Security of Ed25519: Theory and Practice". In: *Symposium on Security and Privacy (SP)* (2021). DOI: 10.1109/SP40001.2021.00042. Also at ia.cr/2020/823 (Cited on pp. 11, 15–17).

[**BCKMTZ22**] Mihir Bellare, Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. "Better than Advertised Security for Non-Interactive Threshold Signatures". In: (2022). Combines two papers: "How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures" at ia.cr/2021/1375, and "Stronger Security for Non-Interactive Threshold Signatures: BLS and FROST" at ia.cr/2022/833. (Cited on p. 33).

[**BDLSY11**] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-Speed High-Security Signatures". In: *Cryptographic Hardware and Embedded Systems — CHES 2011*. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-23951-9_9. Also at Journal of Cryptographic Engineering, vol. 2, pp. 77–89 (2012), 10.1007/s13389-012-0027-1. Also at ia.cr/2011/368 (Cited on pp. 3, 9).

[**BDO14**] Carsten Baum, Ivan Damgård, and Claudio Orlandi. "Publicly Auditable Secure Multi-Party Computation". In: *Security and Cryptography for Networks*. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-10879-7_11. Also at ia.cr/2014/075 (Cited on p. 37).

[**Bea96**] Donald Beaver. "Correlated Pseudorandomness and the Complexity of Private Computations". In: *Proc. 28th Annual ACM Symposium on Theory of Computing*. STOC '96. Association for Computing Machinery, 1996. DOI: 10.1145/237814.237996 (Cited on p. 40).

[**Ber15**] Daniel J Bernstein. "Multi-user Schnorr security, revisited". In: *Cryptology ePrint Archive, Report ia.cr/2015/996* (2015) (Cited on p. 11).

[**BH19**] Joachim Breitner and Nadia Heninger. "Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies". In: *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 1822, 2019, Revised Selected Papers*. Springer. Springer-Verlag, 2019. DOI: 10.1007/978-3-030-32101-7_1. Also at ia.cr/2019/023 (Cited on p. 19).

[**BJLSY15**] Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. *EdDSA for more curves*. https://ed25519.cr.yp.to/eddsa-20150704.pdf. July 2015. Also at ia.cr/2015/677 (Cited on p. 9).

[**BKR00**] Mihir Bellare, Joe Kilian, and Phillip Rogaway. "The Security of the Cipher Block Chaining Message Authentication Code". In: *J. Comput. Syst. Sci.* 61.3 (December 2000). DOI: 10.1006/jcss.1999.1694. Earlier version at CRYPTO 1994, LNCS vol. 839, DOI:10.1007/3-540-48658-5_32 (Cited on p. 16).

[**Bla79**] G. R. Blakley. "Safeguarding cryptographic keys". In: *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society, June 1979. DOI: 10.1109/AFIPS.1979.98 (Cited on p. 3).

[**Ble00**] Daniel Bleichenbacher. "On the generation of one-time keys in DL signature schemes". In: *Presentation at IEEE P1363 working group meeting*. 2000 (Cited on p. 19).

[**BLLOR21**] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. "On the (in)security of ROS". In: *Advances in Cryptology — EUROCRYPT 2021*. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-77870-5_2. Also at ia.cr/2020/945 (Cited on p. 32).

[**bmss10**] bushing, marcan, segher, and sven. "Console hacking 2010 ps3 epic fail". In: *27th Chaos Communication Congress*. https://fahrplan.events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf (Accessed March 2022). Chaos Computer Club. 2010 (Cited on p. 19).

[**BN06**] Mihir Bellare and Gregory Neven. "Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma". In: *Proc. 13th ACM Conference on Computer and Communications Security*. CCS '06. Association for Computing Machinery, 2006. DOI: 10.1145/1180405.1180453 (Cited on p. 32).

[**BN08**] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm". In: *J. Cryptol.* 21.4 (September 2008). DOI: 10.1007/s00145-008-9026-x. Also at ia.cr/2000/025 (Cited on p. 16).

[**BST21**] Charlotte Bonte, Nigel P. Smart, and Titouan Tanguy. "Thresholdizing HashEdDSA: MPC to the rescue". In: *International Journal of Information Security* 20 (2021). DOI: 10.1007/s10207-021-00539-6. Also at ia.cr/2020/214 (Cited on pp. 26, 27, 32).

[**BTZ22**] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. "Stronger Security for Non-Interactive Threshold Signatures: BLS and FROST". In: *Cryptology ePrint Archive, Report ia.cr/2022/833* (June 2022) (Cited on p. 33).

[**BV96**] Dan Boneh and Ramarathnam Venkatesan. "Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes". In: *Advances in Cryptology — CRYPTO '96*. Springer. Springer Berlin Heidelberg, 1996. DOI: 10.1007/3-540-68697-5_11 (Cited on p. 19).

[**Call2021a**] Luís Brandão. *Call 2021a for Feedback on Criteria for Threshold Schemes*. https://csrc.nist.gov/projects/threshold-cryptography. June 2021 (Cited on p. 46).

[**Can01**] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proc. 42nd IEEE Symposium on Foundations of Computer Science*. 2001. DOI: 10.1109/SFCS.2001.959888. Extended version at "Universally Composable Security", Journal of the ACM, Vol. 67, Issue 5, October 2020 Art. 28, doi:10.1145/3402457. Also at ia.cr/2000/067 (Cited on p. 35).

[**CD95**] Ronald Cramer and Ivan Damgård. "Secure Signature Schemes based on Interactive Protocols". In: *Advances in Cryptology — CRYPTO' 95*. Springer Berlin Heidelberg, 1995. DOI: 10.1007/3-540-44750-4_24. Also at BRICS Report Series, 1(29), 1994, DOI:10.7146/brics.v1i29.21637 (Cited on p. 15).

[**CDDIM01**] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. "On Adaptive vs. Non-adaptive Security of Multiparty Protocols". In: *Advances in Cryptology — EUROCRYPT 2001*. Springer Berlin Heidelberg, 2001. DOI: 10.1007/3-540-44987-6_17. Also at ia.cr/2001/017 (Cited on p. 35).

[**CFGN96**] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. "Adaptively Secure Multi-Party Computation". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Association for Computing Machinery, 1996. DOI: 10.1145/237814.238015 (Cited on p. 35).

[**CGMA85**] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults". In: *Proc. 26th Annual Symposium on Foundations of Computer Science*. SFCS '85. IEEE Computer Society, 1985. DOI: 10.1109/SFCS.1985.64 (Cited on p. 23).

[**CGN20**] Konstantinos Chalkias, François Garillot, and Valeria Nikolaenko. "Taming the Many EdDSAs". In: *International Conference on Security Standardisation Research*. Springer, 2020. DOI: 10.1007/978-3-030-64357-7_4. Also at ia.cr/2020/1244 (Cited on pp. 15, 17).

[**CKM21**] Elizabeth Crites, Chelsea Komlo, and Mary Maller. *How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures*. Cryptology ePrint Archive, Report ia.cr/2021/1375. 2021 (Cited on pp. 29, 30).

[**DEFKLNS19**] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. "On the Security of Two-Round Multi-Signatures". In: *2019 IEEE Symposium on Security and Privacy (SP)* (2019). DOI: 10.1109/SP.2019.00050. Also at ia.cr/2018/417 (Cited on pp. 25, 29, 30, 32, 41).

[DER21] Ivan Damgård, Daniel Escudero, and Divya Ravi. "Information-Theoretically Secure MPC Against Mixed Dynamic Adversaries". In: *TCC 2021: Theory of Cryptography (19th international conference)*. Springer-Verlag, 2021. DOI: 10.1007/978-3-030-90459-3_20. Also at ia.cr/2021/1163 (Cited on p. 40).

[Des88] Yvo Desmedt. "Society and Group Oriented Cryptography: a New Concept". In: *Advances in Cryptology — CRYPTO '87*. Springer Berlin Heidelberg, 1988. DOI: 10.1007/3-540-48184-2_8 (Cited on p. 3).

[DF90] Yvo Desmedt and Yair Frankel. "Threshold cryptosystems". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, 1990. DOI: 10.1007/0-387-34805-0_28 (Cited on p. 3).

[DH76] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976). DOI: 10.1109/TIT.1976.1055638 (Cited on p. 3).

[Fel87] Paul Feldman. "A Practical Scheme for Non-Interactive Verifiable Secret Sharing". In: *Proc. 28th Annual Symposium on Foundations of Computer Science*. SFCS '87. IEEE Computer Society, 1987. DOI: 10.1109/SFCS.1987.4 (Cited on p. 23).

[FF13] Marc Fischlin and Nils Fleischhacker. "Limitations of the Meta-reduction Technique: The Case of Schnorr Signatures". In: *Advances in Cryptology — EUROCRYPT 2013*. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-38348-9_27. Also at ia.cr/2013/140 (Cited on p. 16).

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli Maurer. "Trading correctness for privacy in unconditional multi-party computation". In: *Advances in Cryptology — CRYPTO '98*. Springer Berlin Heidelberg, 1998. DOI: 10.1007/BFb0055724 (Cited on p. 40).

[FIPS 186-5 (Draft)] National Institute of Standards and Technology (2019). *Digital Signature Standard (DSS).* (U.S. Department of Commerce, Washington, D.C.) Draft Federal Information Processing Standards Publication (FIPS PUBS) 186-5. October 2019. DOI: 10.6028/NIST.FIPS.186-5-Draft.

[FS87] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Springer Berlin Heidelberg, 1987. DOI: 10.1007/3-540-47721-7_12 (Cited on p. 10).

[GJKR99] Rosario Gennaro, Stanisaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Advances in Cryptology — EUROCRYPT'99*. Springer-Verlag, 1999. DOI: 10.1007/3-540-48910-X_21. See also J. Cryptology 20, pp. 51–83, 2007, DOI:10.1007/s00145-006-0347-3 (Cited on p. 24).

[GKMN21] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. "Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions". In: *Advances in Cryptology — CRYPTO 2021*. Springer. 2021. DOI: 10.1007/978-3-030-84242-0_6. Also at ia.cr/2021/1055 (Cited on pp. 26–28, 32).

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM Journal on Computing* 17.2 (1988). DOI: 10.1137/0217017 (Cited on p. 15).

[**HLM21**] Martin Hirt, Chen-Da Liu-Zhang, and Ueli Maurer. "Adaptive Security of Multi-party Protocols, Revisited". In: *TCC 2021: Theory of Cryptography (19th international conference)*. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-90459-3_23. Also at ia.cr/2021/1175 (Cited on p. 36).

[**HM20**] Martin Hirt and Marta Mularczyk. "Efficient MPC with a Mixed Adversary". In: *1st Conference on Information-Theoretic Cryptography (ITC 2020)*. Vol. 163. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. DOI: 10.4230/LIPIcs.ITC.2020.3. Also at ia.cr/2020/356 (Cited on p. 40).

[**HS01**] Nick A Howgrave-Graham and Nigel P. Smart. "Lattice attacks on digital signature schemes". In: *Designs, Codes and Cryptography* 23.3 (2001). DOI: 10.1023/A:1011214926272 (Cited on p. 20).

[**IKMOP13**] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. "On the Power of Correlated Randomness in Secure Computation". In: *Proc. 10th Theory of Cryptography Conference*. TCC'13. Springer-Verlag, 2013. DOI: 10.1007/978-3-642-36594-2_34. Also at https://www.iacr.org/archive/tcc2013/77850598/77850598.pdf (Cited on pp. 40, 41).

[**KG21**] Chelsea Komlo and Ian Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: (2021). DOI: 10.1007/978-3-030-81652-0_2. Also at ia.cr/2020/852 (Cited on pp. 29, 30, 32).

[**KMP16**] Eike Kiltz, Daniel Masny, and Jiaxin Pan. "Optimal Security Proofs for Signatures from Identification Schemes". In: *Advances in Cryptology — CRYPTO 2016*. Springer Berlin Heidelberg, 2016. DOI: 10.1007/978-3-662-53008-5_2. Also at ia.cr/2016/191 (Cited on p. 16).

[**Lin22**] Yehuda Lindell. *Simple Three-Round Multiparty Schnorr Signing with Full Simulatability*. Cryptology ePrint Archive Report ia.cr/2022/374. 2022 (Cited on pp. 28, 32).

[**MPSW19**] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. "Simple Schnorr multi-signatures with applications to bitcoin". In: *Designs, Codes and Cryptography* 87.9 (2019). DOI: 10.1007/s10623-019-00608-x. Also at ia.cr/2018/068 (Cited on pp. 26, 36, 41).

[**MTR22**] John Preuß Mattsson, Erik Thormarker, and Sini Ruohomaa. *Deterministic ECDSA and EdDSA Signatures with Additional Randomness*. Internet-Draft. https://datatracker.ietf.org/doc/html/draft-mattsson-cfrg-det-sigs-with-noise-04. Internet Engineering Task Force, February 2022 (Cited on p. 20).

[**NISTIR 8214A**] Luís T. A. N. Brandão, Michael Davidson, and Apostol Vassilev. *NIST Roadmap Toward Criteria for Threshold Schemes for Cryptographic Primitives*. NISTIR 8214A, National Institute of Standards and Technology (NIST). July 2020. DOI: 10.6028/NIST.IR.8214A (Cited on p. 37).

[**NKDM03**] Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. "Proactive Two-Party Signatures for User Authentication". In: *Proc. Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*. The Internet Society, 2003. Available at https://www.ndss-symposium.org/ndss2003 (Cited on p. 28).

[**NRS21**] Jonas Nick, Tim Ruffing, and Yannick Seurin. "MuSig2: Simple Two-Round Schnorr Multi-signatures". In: *Advances in Cryptology — CRYPTO 2021*. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-84242-0_8. Also at ia.cr/2020/1261 (Cited on pp. 29, 30, 32).

[**NRSW20**] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. "MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces". In: *Proc. 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Association for Computing Machinery, 2020. DOI: 10.1145/3372297.3417236. Also at ia.cr/2020/1057 (Cited on pp. 26–28).

[**Ped91**] Torben Pryds Pedersen. "A Threshold Cryptosystem without a Trusted Party". In: *Advances in Cryptology — EUROCRYPT '91*. Springer Berlin Heidelberg, 1991. DOI: 10.1007/3-540-46416-6_47 (Cited on p. 24).

[**Ped92**] Torben Pryds Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology — CRYPTO '91*. Springer Berlin Heidelberg, 1992. DOI: 10.1007/3-540-46766-1_9 (Cited on p. 24).

[**PS00**] David Pointcheval and Jacques Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *J. Cryptology* 13.3 (January 2000). DOI: 10.1007/s001450010003. Earlier version at Eurocrypt 1996 (doi:10.1007/3-540-68339-9_33) (Cited on p. 16).

[**PS96**] David Pointcheval and Jacques Stern. "Security Proofs for Signature Schemes". In: *Advances in Cryptology — EUROCRYPT '96*. Springer Berlin Heidelberg, 1996. DOI: 10.1007/3-540-68339-9_33 (Cited on p. 16).

[**PSSLR18**] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. "Attacking deterministic signature schemes using fault attacks". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018. DOI: 10.1109/EuroSP.2018.00031. Also at ia.cr/2017/1014 (Cited on p. 20).

[**RFC 8032**] S. Josefsson and I. Liusvaara. "Edwards-Curve Digital Signature Algorithm (EdDSA)". In: *RFC 8032*. Request for Comments (January 2017). Errata exists. DOI: 10.17487/RFC8032 (Cited on p. 3).

[**RP17**] Yolan Romailler and Sylvain Pelissier. "Practical fault attack against the Ed25519 and EdDSA signature schemes". In: *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2017. DOI: 10.1109/FDTC.2017.12 (Cited on p. 20).

[**RRJSS22**] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. *ROAST: Robust Asynchronous Schnorr Threshold Signatures*. Cryptology ePrint Archive Report ia.cr/2022/550. 2022 (Cited on p. 33).

[**RS21**] Lior Rotem and Gil Segev. "Tighter Security for Schnorr Identification and Signatures: A High-Moment Forking Lemma for Σ-Protocols". In: *Advances in Cryptology — CRYPTO 2021*. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-84242-0_9. Also at ia.cr/2021/971 (Cited on p. 16).

[**SB18**] Niels Samwel and Lejla Batina. "Practical fault injection on deterministic signatures: the case of EdDSA". In: *Progress in Cryptology — AFRICACRYPT 2018*. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-89339-6_17 (Cited on p. 20).

[**SBBDS18**] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. “Breaking Ed25519 in WolfSSL”. In: *Topics in Cryptology — Cryptographers’ Track at the RSA Conference (CT-RSA 2018)*. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-76953-0_1. Also at ia.cr/2017/985 (Cited on p. 20).

[**Sch90**] C. P. Schnorr. “Efficient Identification and Signatures for Smart Cards”. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Springer New York, 1990. DOI: 10.1007/0-387-34805-0_22. See also J. Cryptology 4, pp. 161–174, 1991, DOI:10.1007/BF00196725 (Cited on pp. 3, 9–11).

[**Sch99**] Berry Schoenmakers. “A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting”. In: *Advances in Cryptology — CRYPTO’ 99*. Springer Berlin Heidelberg, 1999. DOI: 10.1007/3-540-48405-1_10 (Cited on p. 37).

[**Sha79**] Adi Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (November 1979). DOI: 10.1145/359168.359176 (Cited on pp. 3, 21).

[**Sim94**] Gustavus J. Simmons. “Subliminal Communication is Easy Using the DSA”. In: *Advances in Cryptology — EUROCRYPT ’93*. Springer Berlin Heidelberg, 1994. DOI: 10.1007/3-540-48285-7_18 (Cited on p. 36).

[**SP 800-186**] Lily Chen, Dustin Moody, Andrew Regenscheid, and Karen Randall. *Draft SP 800-186, Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters*. National Institute of Standards and Technology (NIST). October 2019. DOI: 10.6028/NIST.SP.800-186.

[**SP 800-57-P1-R5**] Elaine Barker. *SP 800-57 Part 1 Rev. 5, Recommendation for Key Management: Part 1 — General*. National Institute of Standards and Technology (NIST). May 2022. DOI: 10.6028/NIST.SP.800-57pt1r5 (Cited on p. 18).

[**SS01**] Douglas R. Stinson and Reto Strobl. “Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates”. In: *Information Security and Privacy*. ACISP 2001. Springer Berlin Heidelberg, 2001. DOI: 10.1007/3-540-47719-5_33 (Cited on pp. 24, 28, 36, 41).

[**TTA18**] Akira Takahashi, Mehdi Tibouchi, and Masayuki Abe. “New Bleichenbacher records: Fault attacks on qDSA signatures”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES ’18)* 3 (2018). DOI: 10.13154/tches.v2018.i3.331-371. Also at ia.cr/2018/396 (Cited on p. 19).

[**Wag02**] David Wagner. “A generalized birthday problem”. In: *Advances in Cryptology — CRYPTO 2002*. Springer Berlin Heidelberg, 2002. DOI: 10.1007/3-540-45708-9_19. Also at https://www.iacr.org/archive/crypto2002/24420288/24420288.pdf (Cited on p. 30).

[**WNR20**] Pieter Wuille, Jonas Nick, and Tim Ruffing. “BIP 340: Schnorr Signatures for secp256k1”. In: *Bitcoin Improvement Proposals*. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki. GitHub, January 2020 (Cited on p. 10).