Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Robust Threshold Schnorr Signatures

Διπλωματικη Εργασια

του

ΚΟΓΙΟΥ Λ. ΔΗΜΗΤΡΙΟΥ

Επιβλέπων: Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Robust Threshold Schnorr Signatures

Διπλωματικη Εργασια

του

ΚΟΓΙΟΥ Λ. ΔΗΜΗΤΡΙΟΥ

Επιβλέπων: Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30 Ιουνίου 2025.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Αριστείδης Παγουρτζής	Νικόλαος Λεονάρδος	Δημήτριος Φωτάκης
Καθηγητής Ε.Μ.Π	Επίκουρος Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Δημήτριος Κόγιος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

1 Ιουνίου 2025

Περίληψη

Οι ψηφιακές υπογραφές είναι κρυπτογραφικοί μηχανισμοί που παρέχουν έναν τρόπο επαλήθευσης της γνησιότητας, της ακεραιότητας και της προέλευσης ενός ψηφιακού μηνύματος. Σε αντίθεση με το μονομελές περιβάλλον, οι πολυ-υπογραφές και οι υπογραφές κατωφλίου απαιτούν συνεργασία μεταξύ πολλών υπογραφόντων καθένας από τους οποίους κατέχει ένα μέρος ενός κοινού ιδιωτικού κλειδιού. Οι πολυ-υπογραφές και οι υπογραφές κατωφλίου είναι επιθυμητές λόγω της αυξημένης ασφάλειας που προσφέρουν σε σύγκριση με τις υπογραφές ενός μόνο χρήστη. Ωστόσο, για να υιοθετηθούν τέτοια σχήματα σε πραγματικές εφαρμογές, είναι ζωτικής σημασίας να ικανοποιούν κάποιες πολύ σημαντικές ιδιότητες που να καλύπτουν τις ανάγκες των εφαρμογών του πραγματικού κόσμου, ενώ ταυτόχρονα να είναι όσο το δυνατόν πιο αποδοτικά, ελαχιστοποιώντας τον όγκο επικοινωνίας και τον υπολογιστικό φόρτο.

Σε αυτήν την διπλωματική εργασία μελετάμε διάφορα σχήματα πολυ-υπογραφών και υπογραφών κατωφλίου, εστιάζοντας στο κομμάτι των υπογραφών Schnorr. Εξετάζουμε απειλές ασφαλείας, ορισμούς ασφαλείας σε διαφορετικά περιβάλλοντα και συγκρίνουμε τα σχήματα μεταξύ τους. Επιπλέον, συζητάμε εφαρμογές και αναλύουμε πραγματικά συστήματα στα οποία χρησιμοποιούνται τέτοια σχήματα. Επιπλέον, σχεδιάζουμε ένα πρωτόκολλο περιτύλιξης (wrapper protocol) που προσθέτει ευρωστία (robustness) (δηλαδή την εγγύηση ότι t έντιμοι υπογράφοντες μπορούν να παράγουν έγκυρη υπογραφή ακόμα και επί παρουσίας κακόβουλων συμμετεχόντων που προσπαθούν να διαταράξουν το πρωτόκολλο) στο προσαρμοστικά ασφαλές (adaptively secure) σχήμα υπογραφών κατωφλίου Sparkle+ στο ασύγχρονο περιβάλλον. Ονομάζουμε αυτό το πρωτόκολλο Sparkling ROAST (βασίζεται στο ROAST, το πρωτόκολλο περιτύλιξης για το σχήμα υπογραφών FROST). Παρουσιάζουμε δύο εκδοχές του Sparkling ROAST: η πρώτη εγγυάται ευρωστία όταν υπάρχουν t έντιμοι υπογράφοντες και απαιτεί το πολύ $O(n^2)$ εσωτερικές συνεδρίες, ενώ η δεύτερη επιτυγχάνει ευρωστία με το πολύ $O(n)$ εσωτερικές συνεδρίες, αλλά απαιτεί $\frac{f\cdot(t-1)}{2} + t$ έντιμους υπογράφοντες αντί για t (όπου f είναι ο αριθμός των κακόβουλων μελών). Τέλος, συζητάμε πώς το Sparkling ROAST μπορεί να επεκταθεί και σε άλλα σχήματα με ακόμη περισσότερους γύρους, αρκεί αυτά τα σχήματα να υποστηρίζουν αναγνωρίσιμες αποχωρήσεις (identifiable aborts).

Λέξεις Κλειδιά

Κρυπτογραφία, Ψηφιακές υπογραφές, Υπογραφές κατωφλίου, Πολυ-υπογραφές, υπογραφές Schnorr, Ευρωστία σε υπογραφές κατωφλίου

Abstract

Digital signatures are cryptographic mechanisms that provide a way to verify the authenticity, integrity and origin of a digital message. Unlike in the single-party setting, multi- and threshold signatures require cooperation among multiple signers each holding a share of a common private key. Multi-signatures allow a group of signers to jointly produce a compact signature, indistinguishable from a single-party one, thereby optimizing space and verification efficiency. Threshold signatures extend this idea, enabling any subset of signers meeting a predefined threshold (symbolized as t) to collaboratively generate a valid signature. Multi- and threshold signatures are desired due to their increased security in contrast to single-party signatures. However, in order for real-world applications to adopt such schemes, it is vital that they follow some very important properties that will cover the needs of real-world applications as well as being as efficient as possible by minimizing the amount of communication and computation needed.

In this thesis, we take a look at multiple multi- and threshold signatures schemes focusing on the domain of Schnorr signatures. We look at security threats, definitions of security in different settings and compare schemes as we go along. We also discuss applications and look at real-world systems where such schemes are used. Moreover, we present a new novel wrapper protocol that adds robustness (i.e the guarantee that t honest signers are able to obtain a valid signature even in the presence of other malicious signers who try to disrupt the protocol) to the adaptively secure Sparkle+ threshold signing scheme in the asynchronous setting. We call our wrapper protocol Sparkling ROAST (it is based on the ROAST wrapper protocol for the FROST signing scheme). We look at two versions on Sparkling ROAST : the first guarantees robustness when t honest signers are present and needs at most $O(n^2)$ internal sessions while the second guarantees robustness in at most $O(n)$ internal sessions but requires $\frac{f\cdot(t-1)}{2} + t$ honest signers instead of t (where f is the number of adversarial members). Finally, we discuss how Sparkling ROAST can be extended to other schemes with even more rounds, as long as said schemes provide identifiable aborts.

Keywords

Cryptography, Digital Signatures, Threshold Signatures, Multi-signatures, Schnorr Signatures, Robustness in threshold signatures

Ευχαριστίες

Θα ήθελα καταρχάς να ευχαριστήσω τον καθηγητή κ. Αριστείδη Παγουρτζή για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα πολύ ενδιαφέρον και ενεργό κομμάτι της κρυπτογραφίας. Ακόμη, ευχαριστώ τους κ. Νικόλαο Λεονάρδο και κ. Δημήτριο Φωτάκη για τη συμμετοχή τους στην τριμελή εξεταστική επιτροπή. Επίσης, θα ήθελα να ευχαριστήσω την υποψήφια διδάκτορα Μαριάννα Σπυράκου και τον Δρ. Παναγιώτη Γροντά για την καθοδήγησή τους και την συνεργασία τους. Είμαι σίγουρος ότι δεν θα μπορούσα να πετύχω τα αποτελέσματα αυτής της εργασίας χωρίς την ώθησή τους. Φυσικά, οφείλω ένα τεράστιο ευχαριστώ στους γονείς μου και στις αδελφές μου που με στήριζαν και συνεχίζουν να με στηρίζουν καθ' όλη τη διάρκεια της σταδιοδρομίας μου. Τέλος, θα ήθελα να ευχαριστήσω κάποια άτομα με τα οποία μοιράστηκα πολλές φοιτητικές αναμνήσεις και περάσαμε αμέτρητες ώρες στα αναγνωστήρια σε κάθε εξεταστική, αυτοί είναι οι ΠΡ, ΔΔ, ΤΑ, ΚΧ, ΚΣ, ΑΣ, ΜΚ, ΓΜ, ΗΠ, ΑΚ, ΗΘ, ΝΚ, ΚΛ, ΓΑ, ΜΖ, ΕΜ.

Αθήνα, Ιούνιος 2025

Δημήτριος Κόγιος

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εκτεταμένη Ελληνική Περίληψη

Στο παρόν κεφάλαιο ακολουθεί μία εκτεταμένη ελληνική παρουσίαση του περιεχομένου αυτής της διπλωματικής εργασίας. Τα υποκεφάλαια έχουν την ίδια δομή με αυτή της αγγλικής εκδοχής και ο αναγνώστης παραπέμπεται στα αντίστοιχα σημεία της για ορισμένες λεπτομέρειες που έχουν παραληφθεί.

1.1 Υπόβαθρο

Ξεκινάμε με μια εισαγωγή στις βασικές έννοιες και στα απαραίτητα εργαλεία της σύγχρονης κρυπτογραφίας που αποτελούν το θεωρητικό υπόβαθρο για τις επόμενες ενότητες.

Η ασφάλεια κάθε κρυπτογραφικού συστήματος καθορίζεται από μια παράμετρο ασφάλειας κ. Για τη συμβολοποίηση χρησιμοποιείται η εναδική (unary) αναπαράσταση 1^κ ως είσοδος σε αλγορίθμους ενώ η τυχαία επιλογή ενός στοιχείου από κάποιο πεπερασμένο σύνολο S συμβολίζεται με $x \xleftarrow{\$} S$. Υποθέτουμε ότι όλοι οι αλγόριθμοι είναι πιθανοτικοί και πολυωνυμικοί (PPT) εκτός αν δηλώνεται διαφορετικά, ενώ αποκαλούμε αμελητέες τις συναρτήσεις που μειώνονται γρηγορότερα από κάθε αντίστροφη πολυωνυμική συνάρτηση.

Κεντρική θέση στην κρυπτογραφία κατέχουν οι συναρτήσεις σύνοψης ή συναρτήσεις κατακερματισμού (hash functions). Αυτές αποτελούν μια απεικόνιση από έναν μεγάλο χωρό τιμών σε έναν μικρότερο και πρέπει να πληρούν συγκεκριμένες ιδιότητες για να θεωρούνται ασφαλείς. Οι βασικές απαιτήσεις είναι η αντίσταση στην εύρεση προεικόνας, δεύτερης προεικόνας και συγκρούσεων. Υπάρχει ιεραρχία μεταξύ αυτών των ιδιοτήτων καθώς η ισχυρότερη (αντίσταση συγκρούσεων) συνεπάγεται τις ασθενέστερες.

Άλλες πολύ σημαντικές έννοιες είναι τα ιδεατά μοντέλα που χρησιμοποιούνται στις αποδείξεις ασφάλειας των κρυπτογραφικών πρωτοκόλλων. Στην παρούσα διπλωματική χρειαζόμαστε μόνο το Μοντέλο Τυχαίου Μαντείου [4] (ROM) και το Μοντέλου των Αλγεβρικών Ομάδων [5] (AGM). Στο πρώτο, θεωρούμε ότι οι συναρτήσεις σύνοψης συμπεριφέρονται σαν προγραμματιζόμενες τυχαίες συναρτήσεις ενώ στο δεύτερο απαιτούμε κάθε αντίπαλος (adversary) που παρουσιάζει κάποιο στοιχείο της ομάδας, να παρουσιάζει και μια αναπαράσταση αυτού του στοιχείου από στοιχεία που έχει ήδη δει προηγουμένως.

Οι ψηφιακές υπογραφές (digital signatures) παρέχουν ισχυρούς μηχανισμούς επαλήθευσης της αυθεντικότητας ενός μηνύματος. Σε ένα σχήμα ψηφιακών υπογραφών, κάθε χρήστης διαθέτει ένα ζεύγος κλειδιών (ένα δημόσιο και ένα ιδιωτικό) και υπογράφει μηνύματα με υπογραφές που μόνο εκείνος μπορεί να δημιουργήσει αλλά οποιοσδήποτε μπορεί να

επαληθεύσει. Λέμε ότι ένα τέτοιο σχήμα έχει την ιδιότητα της ορθότητας (correctness) αν κάθε υπογραφή που δημιουργείται με τίμιο τρόπο γίνεται αποδεκτή από τον αλγόριθμο επαλήθευσης. Πολύ σημαντική ιδιότητα είναι και η ασφάλεια που περιγράφεται μέσω της ιδιότητας αδυναμίας πλαστογράφησης (unforgeability) η οποία εκφράζει την αδυναμία ενός αντιπάλου να παράγει υπογραφές ακόμα και αν έχει στη διάθεσή του υπογραφές άλλων μηνυμάτων της επιλογής του (EUF-CMA). Οι ορισμοί του παιγνίου της ασφάλειας για ψηφιακές υπογραφές φαίνεται στην εικόνα 3.1.

Ένα σχήμα ψηφιακών υπογραφών πάνω στο οποίο έχει γίνει εκτενής έρευνα είναι οι ψηφιακές υπογραφές Schnorr [6] που είναι η μη-διαδραστική εκδοχή του πρωτοκόλλου ταυτοποίησης Schnorr [7] χρησιμοποιώντας την μετατροπή των Fiat-Shamir [8]. Το σχήμα υπογραφής Schnorr βασίζεται σε τρεις φάσεις: τη φάση της δέσμευσης σε ένα τυχαίο στοιχείο (commitment), τη φάση της πρόκλησης (challenge) και τη φάση της απόκρισης/απάντησης (response). Η ασφάλεια του σχήματος βασίζεται στη δυσκολία του προβλήματος εύρεσης διακριτού λογαρίθμου στο Μοντέλου του Τυχαίου Μαντείου.

Ένος κλασικός τρόπος για να διαμοιραστεί ένα μυστικό είναι η μέθοδος του Shamir [1] που βασίζεται στην ιδιότητα της παρεμβολής Lagrange ενός πολυωνύμου. Το μυστικό κωδικοποιείται ως ο σταθερός όρος ενός πολυωνύμου. Οποιοδήποτε υποσύνολο επαρκούς μεγέθους μπορεί να ανακατασκευάσει το αρχικό μυστικό με τη βοήθεια των συντελεστών Lagrange.

Η θεωρητική βάσης ασφάλειας όλων των παραπάνω εργαλείων βασίζεται σε υπολογιστικές υποθέσεις. Η Υπόθεση Διακριτού Λογαρίθμου δηλώνει ότι δοθέντος ενός του στοιχείου g^x, είναι υπολογιστικά αδύνατο να ανακτηθεί το x. Μια εκδοχή της προηγούμενης υπόθεσης είναι η Αλγεβρική έκδοση του Προβλήματος του Ενός Επιπλέον Λογαρίθμου (AOMDL) στην οποία ο αντίπαλος έχοντας πρόσβαση σε περιορισμένο αριθμό ερωτήσεων σε μαντείο λογαρίθμων, δεν μπορεί να υπολογίσει περισσότερους λογάριθμους πέρα από αυτούς που έλαβε από το μαντείο. Στις εικόνες 3.3 και 3.4 παρουσιάζονται οι ορισμοί των προβλημάτων στις οποίες βασίζονται οι δύο αυτές υποθέσεις.

1.2 Πολυ-υπογραφές και Υπογραφές Κατωφλίου

Η συνεργατική υπογραφή μηνυμάτων από πολλούς συμμετέχοντες εισάγει ισχυρούς μηχανισμούς ασφάλειας. Συγκεκριμένα, υπάρχουν δύο έννοιες: οι πολυ-υπογραφές (multisignatures) και οι υπογραφές κατωφλίου (threshold signatures). Στις πρώτες, όλα τα μέλη μιας ομάδας οφείλουν να συμμετάσχουν στη διαδικασία υπογραφής για να παραχθεί μία ενιαία υπογραφή. Αντίθετα, στις δεύτερες μόνο ένα υποσύνολο από t μέλη αρκεί για την παραγωγή της υπογραφής ακόμα κι αν ο συνολικός αριθμός μελών της ομάδας είναι n. Στόχος και των δύο μοντέλων είναι η παραγωγή υπογραφών σταθερού μεγέθους, ανεξάρτητα από τον αριθμό των συμμετεχόντων, ώστε να διατηρείται η αποδοτικότητα και η συμβατότητα με τα μονομελή σχήματα υπογραφών.

Η επικοινωνία μεταξύ των συμμετεχόντων σε αυτά τα σχήματα υλοποιείται μέσω αυθεντικοποιημένων καναλιών δηλαδή υποθέτουμε ότι κάθε μήνυμα που αποστέλλεται συνοδεύεται από μια ψηφιακού υπογραφή ενός μονομελούς σχήματος για να διασφαλίζεται η ακεραιότητα και η αυθεντικότητα των μηνυμάτων. Σε ορισμένες περιπτώσεις απαιτείται ένας

ημι-έμπιστος συντονιστής (coordinator) για τη διαχείριση των επικοινωνιών ο οποίος δεν μπορεί να παραβιάσει την ασφάλεια του σχήματος. Καθ'όλη τη διάρκεια της συγκεκριμένης διπλωματικής θεωρούμε ότι κάθε σχήμα θα έχει έναν συντονιστή.

Τα σχήματα πολυ-υπογραφών και υπογραφών κατωφλίου αποτελούνται από ακολουθίες αλγορίθμων που εκτελούνται σε γύρους. Κάθε υπογράφων τρέχει τους αλγορίθμους, ανταλλάσει τα απαραίτητα μηνύματα με τα υπόλοιπα μέλη και διατηρεί εσωτερική κατάσταση με ιδιωτικές πληροφορίες. Όταν ολοκληρωθεί η διαδικασία, τα τελικά μηνύματα που ονομάζονται επιμέρους υπογραφές συνδυάζονται για την παραγωγή της τελικής υπογραφής η οποία επαληθεύεται όπως μια κανονική υπογραφή με βάση το δημόσιο κλειδί της ομάδας.

Για την εγκυρότητα ενός τέτοιου συστήματος απαιτείται ορθότητα: εφόσον όλοι οι συμμετέχοντες ακολουθήσουν τίμια το πρωτόκολλο, η υπογραφή που παράγεται θα είναι έγκυρη. Εξίσου σημαντική είναι η ασφάλεια του σχήματος που προσομοιώνεται με το πείραμα/παίγνιο πλαστογράφησης υπογραφής από αντίπαλο που ελέγχει αριθμό μελών μικρότερο από το απαιτούμενο κατώφλι. Οι ορισμοί των παιγνίων ορθότητας και μη-πλαστογράφησης εμφανίζονται στις εικόνες 4.1 και 4.2 αντίστοιχα.

Στην τρέχουσα διπλωματική εστιάζουμε σε σχήματα πολυ-υπογραφών και υπογραφών κατωφλίου όπου η τελική υπογραφή της ομάδας μοιάζει απόλυτα με τη μονομελή υπογραφή Schnorr. Τέτοια πρωτόκολλα προσφέρουν συμβατότητα με συστήματα επαλήθευσης κλασικού Schnorr. Επίσης, με αυτόν τον τρόπο ένας οποιοσδήποτε επαληθευτής μπορεί να επαληθεύσει την υπογραφή χωρίς να γνωρίζει αν προήλθε από έναν υπογράφοντα ή από ομάδα.

Η χρήση τέτοιων σχημάτων έχει επεκταθεί σημαντικά μετά την υιοθέτηση των υπογραφών Schnorr σε συστήματα όπως το Bitcoin [9] αντικαθιστώντας παλαιότερες μεθόδους όπως το ECDSA. Επιπλέον, οργανισμοί όπως ο NIST έχουν εκφράσει επίσημο ενδιαφέρον για την τυποποίηση των συστημάτων υπογραφών κατωφλίου [10] , [11] γεγονός που υποδηλώνει τη σημασία τους σε μελλοντικές υποδομές. Πρακτικές εφαρμογές τέτοιων τεχνικών περιλαμβάνουν την ασφάλεια ψηφιακών πορτοφολιών, τη χρήση τους σε αρχές πιστοποίησης, στο πεδίο των γεννητριών τυχαιότητας και σε ηλεκτρονικές ψηφοφορίες.

Τα σχήματα πολυ-υπογραφών και υπογραφών κατωφλίου μπορούν να συγκριθούν μεταξύ τους και να αξιολογηθούν με βάση συγκεκριμένες μετρικές υπολογιστικής και επικοινωνιακής πολυπλοκότητας. Οι βασικοί άξονες τους οποίους χρησιμοποιούμε περιλαμβάνουν τον αριθμό των γύρων επικοινωνίας, τον συνολικό όγκο μεταδιδόμενων δεδομένων και τους απαιτούμενους υπολογισμούς ανά υπογράφοντα [12].

1.3 Προκλήσεις ως προς την Ασφάλεια των Πολυ-υπογραφών και Υπογραφών Κατωφλίου

Μία από τις σημαντικότερες προκλήσεις που αντιμετωπίζουν τα συνεργατικά σχήματα Schnorr υπογραφών είναι το πρόβλημα ROS το οποίο έχει συνδεθεί στενά με την ασφάλεια τυφλών υπογραφών [13]. Ο ορισμός του προβλήματος εμφανίζεται στην εξίσωση 5.2. Αν ένας αντίπαλος μπορεί να επιλύσει μία τέτοια εξίσωση, τότε αποκτά τη δυνατότητα να πλαστογραφήσει $\ell+1$ υπογραφές έχοντας επικοινωνήσει μόλις ℓ φορές με τίμιους υπογράφοντες.

Μια βασική τεχνικής είναι ο Γενικευμένος Αλγόριθμος Γενεθλίων του Wagner [14] που γενικεύει το κλασικό πρόβλημα γενεθλίων στο πρόβλημα k-sum δηλαδή της εύρεσης k στοιχείων (το καθένα από μια διαφορετική λίστα) τα οποία αθροίζουν στο μηδέν. Ο αλγόριθμος οργανώνει τα δεδομένα σε δέντρα και πετυχαίνει σημαντική βελτίωση πολυπλοκότητας από άλλες τεχνικές. Η τεχνική αυτή μπορεί να προσαρμοστεί και στο πρόβλημα ROS.

Το βασικό πρόβλημα της επίθεσης του Wagner είναι η εκθετική πολυπλοκότητά της καθιστώντας έτσι την απειλή περιορισμένη. Ωστόσο, στο [15] παρουσιάζεται ένας πολυωνυμικός αλγόριθμος για την επίλυση του ROS ο οποίος ανέτρεψε πολλά σχήματα ασφαλείας. Η βασική ιδέα είναι η χαλάρωση του περιορισμού που έθετε ο αλγόριθμος του Wagner ότι η τελευταία γραμμή του πίνακα πρέπει να είναι σταθερή. Η ευκολία υπολογισμού αυτής της λύσης έθεσε σε αμφισβήτηση ακόμη και αποδείξεις ασφάλειας πρωτοκόλλων που δεν βασίζονταν άμεσα στο πρόβλημα ROS.

Ένα χαρακτηριστικό παράδειγμα είναι το σχήμα υπογραφών που ονομάζουμε InsecureMusig [16] που θεωρούταν ασφαλές μέχρι τότε. Ωστόσο, παρ όλη την απόδειξη ασφλαλείας που είχε, αποδείχθηκε ευάλωτο σε δύο διαφορετικές επιθέσεις. Η πρώτη ήταν η επίθεση Drijvers [17] ενώ η δεύτερη είναι άμεσο αποτέλεσμα της πολυωνυμικής επίθεσης στο ROS. Και οι δύο αυτές επιθέσεις χρησιμοποιούν πολλαπλές παράλληλες συνεδρίες υπογραφής με τους τίμιους υπογράφοντες και εκμεταλλευόμενες την ελευθερία επιλογής της δέσμευσης Schnorr, συνθέτουν μια πλαστή υπογραφή η οποία επαληθεύεται ως έγκυρη.

Η αποκάλυψη αυτών των αδυναμιών αποκάλυψε σοβαρά σφάλματα στην αρχική απόδειξη ασφάλειας του InsecureMusig, η οποία βασιζόταν στην υπόθεση OMDL (One-More Discrete Logarithm). Η μεθοδολογία της απόδειξης επέτρεπε στον αντίπαλο υπερβολικά πολλές προσβάσεις σε μαντείο διακριτού λογάριθμου, καθιστώντας την απόδειξη ανεπαρκή. Επιπλέον, αποδείχθηκε ότι καμία γνωστή τεχνική δεν μπορεί να αποδείξει την ασφάλεια του InsecureMusig υπό τις παραδοσιακές υπολογιστικές υποθέσεις, καθιστώντας το σχήμα πρακτικά ανασφαλές.

1.4 Τα σχήματα MuSig1 και MuSig2 και τεχνικές ασφάλειας για ταυτόχρονες συνεδρίες

Η έλλειψη ασφάλειας του πρωτοκόλλου InsecureMusig σε ταυτόχρονες συνεδρίες οδήγησε στην ανάπτυξη των πρωτοκόλλων MuSig1 και MuSig2 τα οποία επιλύουν το πρόβλημα χωρίς να θυσιάζουν την πρακτικότητα. Το βασικό πρόβλημα του InsecureMusig βρίσκεται στην πλήρη ελευθερία που διαθέτει ο επιτιθέμενος στον προσδιορισμό της ομαδικής δέσμευσης της υπογραφής Schnorr επιτρέποντάς του να προσαρμόζει τις δεσμεύσεις του αφού πρώτα παρατηρήσει τις δεσμεύσεις των τίμιων συμμετεχόντων. Δύο κύριες τεχνικές έχουν προταθεί για την αντιμετώπιση αυτού του προβλήματος: η εισαγωγή επιπλέον γύρου δέσμευσης και η χρήση δεσμευτικών παραγόντων (binding factors) που εξαρτώνται από όλες τις δεσμεύσεις.

Το πρωτόκολλο MuSig1 [18] βασίζεται στην πρώτη τεχνική προσθέτοντας επιπλέον γύρο κατά τον οποία οι συμμετέχοντες αποστέλλουν μία δέσμευση για τη δέσμευση Schnorr πριν την φανερώσουν. Αυτό αποτρέπει επιθέσεις σαν αυτές που εφαρμόστηκαν στο InsecureMusig καθώς οποιαδήποτε ανακολουθία οδηγεί σε ακύρωση της συνεδρίας. Το MuSig1 αποτελεί

μια ασφαλή εκδοχή του InsecureMusig προσθέτοντας έναν έξτρα γύρο αλλά διατηρώντας παρόμοια δομή.

Αντιθέτως, το MuSig2 [19] υιοθετεί την τεχνική των δεσμευτικών παραγόντων επιτρέποντας ασφαλή παραγωγή υπογραφών σε δύο γύρους. Κάθε συμμετέχων δημιουργεί δύο τυχαίες τιμές, και η τελική δέσμευση της ομάδας εξαρτάται από όλες τις δημοσιοποιημένες δεσμεύσεις μέσω ενός καθολικού παράγοντα. Ο παράγοντας αυτός εξασφαλίζει ότι καμία δέσμευση δεν μπορεί να οριστεί εκ των προτέρων, καθιστώντας τις επιθέσεις τύπου ROS μη εφαρμόσιμες. Η χρήση δύο τυχαίων τιμών ανά υπογράφοντα είναι κρίσιμη, καθώς η χρήση μίας μόνο τυχαίας τιμής θα επέτρεπε στον επιτιθέμενο να εξαλείψει της επίδραση του δεσμευτικού παράγοντα.

Στην τελική σύγκριση, το MuSig2 υπερτερεί ως προς την πρακτικότητα, καθώς απαιτεί λιγότερους γύρους αλληλεπίδρασης – κρίσιμο χαρακτηριστικό για αποκεντρωμένα δίκτυα και εφαρμογές με περιορισμένο αριθμό μηνυμάτων. Η επιλογή μεταξύ των δύο εξαρτάται από τις απαιτήσεις του συστήματος ως προς την ασφάλεια, το latency και τους υπολογιστικούς πόρους των μερών που συμμετέχουν. Στα σχήματα 6.1 και 6.2 περιγράφονται με λεπτομέρεια οι αλγόριθμοι των δύο σχημάτων.

1.5 Κατανεμημένη Παραγωγή Κλειδιών για Υπογραφές Κατωφλίου

Η βασική διαφορά μεταξύ των πολυ-υπογραφών και των υπογραφών κατωφλίου είναι ότι στα συστήματα των πρώτων απαιτείται η συμβολή όλων των μελών ενώ στις υπογραφές κατωφλίου αρκεί η συμμετοχή t από τα συνολικά n μέλη. Αυτό ωστόσο. εισάγει την ανάγκη για ειδικές τεχνικές παραγωγής κλειδιών ώστε κάθε υποσύνολο t αξιόπιστων μελών να μπορεί να παραγάγει έγκυρη υπογραφή που να επαληθεύεται από το μοναδικό δημόσιο κλειδί της ομάδας.

Για την υλοποίηση αυτής της ιδιότητας, χρησιμοποιούνται πρωτόκολλα διαμοιρασμού μυστικών (secret sharing), με γνωστότερο το Shamir Secret Sharing [1]. Σε αυτό το σχήμα, ένα μυστικό μοιράζεται σε n συμμετέχοντες μέσω ενός πολυωνύμου βαθμού $t-1$ και έτσι, κάθε υποσύνολο μεγέθους t μπορεί να το ανακατασκευάσει. Ωστόσο, το Shamir Secret Sharing απαιτεί έναν έμπιστο διαχειριστή (trusted dealer) κάτι μη επιθυμητό. Για να αποφευχθεί η εμπιστοσύνη στον διαχειριστή προτείνονται και περιγράφονται σχήματα όπως το Feldman's VSS και το Pedersen's VSS που προσθέτουν μηχανισμούς επαλήθευσης της ορθότητας των επιμέρους μεριδίων ώστε κάθε χρήστης να μπορεί να βεβαιωθεί ότι έλαβε το σωστό μερίδιο από τον διαχειριστή.

Χρησιμοποιώντας πρωτόκολλα διαμοιρασμού, παρουσιάζουμε τον αλγόριθμο DKG του Pedersen [20]. Σε αυτόν τον αλγόριθμο, ο καθένας από τους n συμμετέχοντες ενεργεί ως dealer τρέχοντας το δικό του VSS. Οι τελικές τιμές των μεριδίων και το κοινό μυστικό προκύπτουν από την άθροιση των τιμών όλων των επιμέρους VSS. Το πρωτόκολλο PedPoP [21] αποτελεί μια επέκταση του Pedersen DKG προσθέτοντας αποδείξεις κατοχής (proof of pocession) για την αποτροπή επιθέσεων rogue-key. Και τα δύο αυτά πρωτόκολλα απαιτούν δύο γύρους. Τα πρωτόκολλα περιγράφονται στην εικόνα 7.2.

Παρόλο που ο αλγόριθμος Pedersen DKG είναι λειτουργικός και αποδοτικός, δεν βασίζει την ασφάλεια του σε κάποια υπόθεση. Αυτό ακριβώς εκμεταλλεύτηκαν οι Gennaro et al. [22] εντοπίζοντας μια επίθεση επιρροής κατά την οποία ένας κακόβουλος συμμετέχων μπορεί να επηρεάσει την κατανομή του τελικού δημόσιου κλειδιού παραβιάζοντας έτσι την ιδιότητα της ομοιόμορφης κατανομής. Η επίθεση αυτή βασίζεται στην ικανότητα του επιτιθέμενου να αποκλείει συμμετέχοντες αφότου έχει παρατηρήσεις τις δεσμεύσεις τους. Για την επίλυση του προβλήματος, οι προηγούμενοι συγγραφείς προτείνουν μια τροποποιημένη εκδοχή του Pedersen DKG που χρησιμοποιεί το Pedersen VSS για να αποκρύψει το δημόσιο κλειδί κατά τη διάρκεια του πρωτοκόλλου.

Τέλος κοιτούμε μία καινούργια εργασία [23] που καταπιάνεται με τον ορισμό ιδιοτήτων ασφάλειας για τους αλγόριθμους DKG και ορίζει δομικά στοιχεία που προσφέρουν αυτές τις ιδιότητες. Κάποιος που θέλει να δημιουργήσει έναν νέο αλγόριθμο DKG αρκεί να χρησιμοποιήσει τα δικά του επιθυμητά δομικά στοιχεία που ακολουθούν κάποιες ιδιότητες. Η ασφάλεια του τελικού DKG προέρχεται απευθείας από την ασφάλεια των δομικών στοιχείων. Αυτή η κατασκευή φαίνεται στην εικόνα 7.4.

1.6 Τα σχήματα υπογραφών κατωφλίου FROST και FROST2

Στη συνέχεια επικεντρωνόμαστε στα σχήματα υπογραφών κατωφλίου FROST [21] και FROST2 [24]. Τα σχήματα της οικογένειας FROST αποτελούνται από δύο γύρους όπου ο πρώτος γύρος είναι ανεξάρτητος του μηνύματος προς υπογραφή και συνεπώς μπορεί να προεπεξεργαστεί προτού ξεκινήσει κάποια συνεδρία.

Η βασική αρχή πίσω από τα δύο σχήματα είναι η εξασφάλιση ότι t από τα n μέλη μπορούν να συνεισφέρουν επιμέρους υπογραφές για να παραχθεί μια έγκυρη ομαδική υπογραφή. Ωστόσο, η μη αποστολή επιμέρους υπογραφής από κάποιο μέλος ή η αποστολή λανθασμένης υπογραφής οδηγεί σε αποτυχία της συνεδρίας. Αυτό καθιστά τα σχήματα FROST μη εύρωστα (non-robust) πράγμα που σημαίνει ότι σε περίπτωση συμπεριφοράς που αποκλίνει από το πρωτόκολλο, η συνεδρία πρέπει να επανεκκινήσει.

Το πρωτόκολλο FROST2 αποτελεί παραλλαγή του αρχικού FROST με βελτιστοποίηση στην πολυπλοκότητα υπολογισμών. Η κύρια διαφορά είναι ότι το FROST2 χρησιμοποιεί έναν κοινό δεσμευτικό παράγοντα αντί για ξεχωριστούς για κάθε συμμετέχοντα (πράγμα που συμβαίνει στο FROST). Η ανάλυση πολυπλοκότητας που διεξάγουμε αποδεικνύει ότι το FROST2 επιτυγχάνει μικρότερη υπολογιστική επιβάρυνση ανά συμμετέχοντα, μειώνοντας τον αριθμό των υψώσεων σε δύναμη που είναι η πιο κοστοβόρα πράξη. Τα δύο σχήματα παρουσιάζονται στην εικόνα 8.1.

Στη συνέχεια αναλύουμε την έννοια της ευρωστίας (robustness) σε υπογραφές κατωφλίου. Ένα εύρωστο σχήμα πρέπει να εγγυάται την παραγωγή υπογραφής όταν υπάρχουν τουλάχιστον t τίμιοι συμμετέχοντες, ακόμα και αν οι υπόλοιποι προσπαθούν να διακόψουν τη διαδικασία. Στο ασύγχρονο μοντέλο επικοινωνίας (π.χ στο διαδίκτυο), όπου δεν υπάρχει εγγύηση χρονικής απόκρισης, η ευρωστία δεν είναι αυτονόητη. Για αυτόν τον λόγο δημιουργήθηκε το πρωτόκολλο ROAST [25] το οποίο αποτελεί ένα πρωτόκολλο "περιτύλιγμα" γύρω από το FROST και προσφέρει ευρωστία στο ασύγχρονο περιβάλλον. Το ROAST τρέχει παράλληλες συνεδρίες FROST στις οποίες κάθε συμμετέχων συνοδεύει την επιμέρους υπογραφή

του με ένα νέο ζεύγος προκαθορισμένων δεσμεύσεων (presignature share). Ο συντονιστής διατηρεί λίστες με τους "αποκριθέντες" και "μη αποκριθέντες" συμμετέχοντες και ξεκινά νέες συνεδρίες με όσους έχουν ανταποκριθεί. Το ROAST εξασφαλίζει ότι μετά από το πολύ $n - t + 1$ συνεδρίες FROST, μία τουλάχιστον θα επιτύχει και συνεπώς μια ομαδική υπογραφή θα δημιουργηθεί. Η κεντρική εγγύηση της ευρωστίας του ROAST βασίζεται στην ιδιότητα της αναγνωρίσιμης αποχώρησης (identifiable abort) που παρέχεται από το FROST, δηλαδή την ικανότητα του συστήματος να εντοπίζει με ακρίβεια ποιος συμμετέχων συμπεριφέρεται κακόβουλα (στέλνοντας λανθασμένη επιμέρους υπογραφή).

1.7 Έννοιες ασφάλειας για Υπογραφές Κατωφλίου

Όπως προαναφέραμε, η βασική έννοια ασφάλειας για όλα τα σχήματα ψηφιακών υπογραφών είναι η έννοια της αδυναμίας πλαστογράφησης (unforgeability). Στα σχήματα υπογραφών κατωφλίου όπου υπάρχει επικοινωνία μεταξύ των μελών, η ερμηνεία της αδυναμίας πλαστογράφησης είναι ότι ένας αντίπαλος, δεν μπορεί να παραγάγει υπογραφές ακόμα και αν ελέγχει συνολικά $t - 1$ υπογράφοντες. Αυτή η έννοια πρόκειται για την απαραίτητη βασική απόδειξη που οφείλει να έχει ένα σχήμα υπογραφών κατωφλίου ώστε να θεωρείται ασφαλές, δεν είναι όμως η μόνη. Σε αυτό το κεφάλαιο εστιάζουμε σε πληρέστερες και πιο ακριβείς έννοιες ασφαλείας και συζητούμε τι είδους ασφάλεια πετυχαίνουν τα προαναφερθέντα σχήματα με βάση αυτές τις έννοιες.

Η βασική εργασία στην οποία βασιζόμαστε είναι η [26] που σκοπό είχε να παρουσιάσει διαφορετικά επίπεδα ασφαλείας για τα μερικώς μη διαδραστικά σχήματα υπογραφών κατωφλίου. Αυτά είναι σχήματα δύο γύρων όπου ο πρώτος γύρος μπορεί να εκτελεστεί προτού ξεκινήσει μια συνεδρία μιας και είναι ανεξάρτητος του μηνύματος. Ειδικότερα, τα σχήματα αυτά χωρίζονται στις παρακάτω φάσεις: παραγωγή προκαθορισμένων δεσμεύσεων (presignatures) από τους συμμετέχοντες, εκκίνηση της συνεδρίας από τον συντονιστή με μία αίτηση, αποστολή επιμέρους υπογραφών από τους υπογράφοντες και συνένωση των υπογραφών σε μία τελική υπογραφή από τον συντονιστή.

Στη συνέχεια παρουσιάζεται η ιεραρχία πέντε επιπέδων για την έννοια της μη πλαστογράφησης όπως προτάθηκε από τους Bellare et al [26]. Τα επίπεδα αυτά (TS-UF-0 έως TS-UF-4) βασίζονται στον αριθμό και την ποσότητα της συμμετοχής των τίμιων συμμετεχόντων σε μια συνεδρία υπογραφής. Σε χαμηλότερα επίπεδα (π.χ TS-UF-0) μια υπογραφή θεωρείται ασήμαντη (trivial) εφόσον ο αντίπαλος απέκτησε έστω και μία επιμέρους υπογραφή για το μήνυμα (αυτό το επίπεδο είναι ισοδύναμο με την περίπτωση όπου ο αντίπαλος ελέγχει $t - 1$ υπογράφοντες) ενώ σε ανώτερα επίπεδα επιτρέπεται μεγαλύτερη ευελιξία στον τρόπο με τον οποίο ο αντίπαλος μπορεί να αναμιγνύει αιτήματα από διάφορες συνεδρίες.

Πέρα όμως από τις έννοιες της απλής μη πλαστογράφησης, υπάρχει και η έννοια της ισχυρής μη πλαστογράφησης (strong unforgeability) η οποία απαιτεί ότι ακόμα και αν ο αντίπαλος έχει λάβει έγκυρη υπογραφή για κάποιο μήνυμα, δεν μπορεί να να δημιουργήσει νέα, διαφορετική υπογραφή για το ίδιο μήνυμα. Στο πλαίσιο των υπογραφών κατωφλίου εξετάζουμε τις διαφορές αυτής της έννοιας έναντι στα μονομελή σχήματα αλλά και τις προκλήσεις που προκύπτουν προσπαθώντας να ορίσουμε αυτήν την έννοια. Τέλος, παρουσιάζουμε τον ορισμό αυτής της έννοιας για υπογραφές κατωφλίου όπως ορίζεται στο [27] όπου ο ορισμός

είναι βασισμένος στις έννοιες ασφαλείας για τυφλές υπογραφές (blind signatures). Πιο συγκεκριμένα, βασίζεται στην προσέγγιση της ¨μίας ακόμας μη πλαστογράφησης¨ (one-more unforgeability) η οποία ζητά από τον αντίπαλο να δημιουργήσει περισσότερες υπογραφές από τις συνεδρίες που εκτελέστηκαν.

Η έννοια της ισχυρής μη πλαστογράφησης για υπογραφές κατωφλίου δεν είναι ανεξάρτητη από τα προηγούμενα επίπεδα ασφαλείας αφού αποδεικνύεται στο [26] ότι συνεπάγεται την ασφάλεια επιπέδου TS-SUF-2. Στην εικόνα 9.3 παρουσιάζονται τα πέντε παίγνια ασφαλείας όπως ορίζονται στο [26] ενώ στην εικόνα 9.5 εμφανίζεται το παίγνιο ισχυρής μη πλαστογράφησης.

1.8 Σχήματα υπογραφών κατωφλίου έναντι Προσαρμοστικών Επιτιθέμενων

Συνεχίζοντας, συγκεντρωνόμαστε στην ασφάλεια των σχημάτων υπογραφών κατωφλίου όταν οι επιτιθέμενοι λειτουργούν προσαρμοστικά (adaptive security), δηλαδή μπορούν να επιλέγουν ανά οποιαδήποτε στιγμή του πρωτοκόλλου ποια μέλη θα διαφθείρουν (φυσικά πάλι υπάρχει ο περιορισμός των $t-1$ συνολικά αντιπάλων). Η μέχρι τώρα ανάλυση της διπλωματικής βασιζόταν σε στατικούς επιτιθέμενους οι οποίοι επιλέγουν τα διεφθαρμένα μέλη εξ αρχής. Ωστόσο, η ασφάλεια ενάντια σε προσαρμοστικούς αντιπάλους είναι πολύ πιο ρεαλιστική ιδιαίτερα σε πραγματικά κατανεμημένα συστήματα, όπως φαίνεται και από τις προδιαγραφές που απαιτούν τα σχήματα για τα οποία ενδιαφέρεται ο NIST [11]. Συγκεκριμένα, σε αυτό το κεφάλαιο περιγράφουμε τις διαφορές της στατικής και προσαρμοστικής ασφάλειας και παρουσιάζουμε δύο σχήματα που πληρούν τις απαιτήσεις της δεύτερης: το Sparkle [28] και το Glacius [29].

Το Sparkle+ [28] ήταν το πρώτο σχήμα Schnorr υπογραφών κατωφλίου με αποδεδειγμένη προσαρμοστική ασφάλεια στο Μοντέλο του Τυχαίου Μαντείου. Η βασική του δομή είναι απλή και θυμίζει το MuSig1 (φυσικά η βασική διαφορά είναι ότι το Sparkle είναι σχήμα υπογραφών κατωφλίου και όχι πολυ-υπογραφών). Εκτελείται σε τρεις γύρους οι οποίοι είναι αποστολή δεσμεύσεων στις τυχαίες τιμές, αποκάλυψη των τυχαίων τιμών που θα χρησιμοποιηθούν στη δέσμευση Schnorr και παραγωγή επιμέρους υπογραφών. Η ασφάλεια του Sparkle εξαρτάται από τον αριθμό των διεφθαρμένων συμμετεχόντων. Στο ROM το σχήμα είναι ασφαλές για έως $t/2$ διαφθορές ενώ αν μεταβούμε στο AGM, το Sparkle είναι πλήρως προσαρμοστικά ασφαλές. Η διαφορά αυτή έγκειται στο γεγονός ότι στην απόδειξη στο ROM είναι η απαραίτητη η χρήση του forking lemma [30] και έτσι δεν μπορούμε να είμαστε σίγουροι ότι ο προσαρμοστικός αντίπαλος θα διαφθείρει τους ίδιους υπογράφοντες στις δύο εκδοχές του πρωτοκόλλου.

Εάν θέλουμε να αποφύγουμε το AGM και να έχουμε ένα πλήρως προσαρμοστικό σχήμα, πρέπει να στρέψουμε την προσοχή μας στο Glacius [29] που ως μειονέκτημα έναντι του Sparkle είναι η αύξηση της πολυπλοκότητας. Το Glacius επιτυγχάνει πλήρης προσαρμοστική ασφάλεια γιατί βασίζει τη δυσκολία στο πρόβλημα απόφασης Diffie-Hellman που είναι μια μη διαδριστική υπόθεση έναντι στο OMDL που χρησιμοποιείται στο Sparkle. Το πρωτόκολλο αυτό αποτελείται από πέντε γύρους και εισάγει επιπλέον ελέγχους συνέπειας μεταξύ

των συμμετεχόντων π.χ έλεγχος συμφωνίας των views του πρωτοκόλλου κάθε συμμετέχοντα. Χαρακτηριστικό του Glacius είναι οι πολλαπλοί γύροι δεσμεύσεων προτού αποκαλυφθούν οι πληροφορίες οι οποίοι υλοποιούνται με αποστολή τιμών συναρτήσεων σύνοψης.

Στην εικόνα 10.3 γίνεται μια σύγκριση των δύο αυτών πρωτοκόλλων με βάσης τις γνωστές μετρικές. Η σύγκριση δείχνει ότι το Sparkle είναι πιο ελαφρύ τόσο σε επικοινωνία όσο και σε υπολογισμό ενώ το Glacius απαιτεί περισσότερους γύρους αλλά προσφέρει καλύτερες εγγυήσεις ασφάλειας.

1.9 Sparkling ROAST: Ένα εύρωστο πρωτόκολλο για το Sparkle+

Στο τελευταίο κύριο κεφάλαιο της διπλωματικής εργασίας, παρουσιάζουμε τη δική μας κατασκευή, το πρωτόκολλο Sparkling ROAST, μία νέα πρόταση που έχει ως στόχο να προσθέσει ευρωστία (robustness) στο τριών γύρων σχήμα υπογραφών κατωφλίου Sparkle+. Σε αντίθεση με το ROAST, το οποίο έχει σχεδιαστεί για σχήματα δύο γύρων όπως το FROST, το Sparkling ROAST είναι ειδικά κατασκευασμένο ώστε να επεκτείνει την έννοια της ευρωστίας σε πιο σύνθετα σχήματα υπογραφών που αποτελούνται από περισσότερους γύρους.

Όπως αναφέραμε όταν παρουσιάζαμε το πρωτόκολλο ROAST, τα σχήματα υπογραφών κατωφλίου εν γένει πάσχουν από την έλλειψη ευρωστίας, δηλαδή της ιδιότητας που ορίζει ότι όσο υπάρχουν τουλάχιστον t τίμιοι υπογράφοντες, το πρωτόκολλο πρέπει να μπορεί να ολοκληρωθεί επιτυχώς. Ειδικότερα, μας ενδιέφερε να προσθέσουμε αυτήν την ιδιότητα στο Sparkle+ στο ασύγχρονο μοντέλο επικοινωνίας όπου (υπενθυμίζουμε) η μόνη εγγύηση είναι ότι τα μηνύματα θα φτάσουν σίγουρα στον προορισμό τους κάποια χρονική στιγμή χωρίς να μπορούμε να περιορίσουμε το χρονικό διάστημα μεταξύ της αποστολής και της άφιξης (latency).

Ο συντονιστής αποτελεί βασικό στοιχείο του Sparkling ROAST καθώς είναι υπεύθυνος για τη διαχείριση πολλαπλών παράλληλων συνεδριών του Sparkle+ παρακολουθώντας την πρόοδο κάθε συμμετέχοντα και διατηρώντας πληροφορίες για την κατάσταση (state) του κάθε συμμετέχοντα. Οι καταστάσεις χωρίζονται σε καταστάσεις ανεξάρτητες των συνεδριών (session-independent) και καταστάσεις ειδικές με τις συνεδρίες (session-specific). Πιο συγκεκριμένα, οι καταστάσεις είναι:

- Responsive : Ο υπογράφων είναι έτοιμος να συμμετάσχει σε μια συνεδρία.
- Blocking2 : Ο υπογράφων δεν έχει αποκαλύψει την προεικόνα της συνάρτησης σύνοψης (preimage) στον δεύτερο γύρο του Sparkle.
- Pending : Ο υπογράφων έχει στείλει την προεικόνα και αναμένει τους υπόλοιπους υπογράφοντες της συνεδρίας να κάνουν το ίδιο.
- Blocking3 : Ο υπογράφων δεν έχει παραδώσει την επιμέρους υπογραφή στον τρίτο γύρο του Sparkle.

Το διάγραμμα καταστάσεων του Sparkling ROAST φαίνεται στην εικόνα 11.2.

Όπως και στο ROAST, έτσι και στο Sparkling ROAST, κεντρική τεχνική είναι το piggybacking δηλαδή η αποστολή νέων presignatures (στην περίπτωση του Sparkle τα presignatures είναι τιμές σύνοψης) για μελλοντικές συνεδρίες έτσι ώστε να εξασφαλίζεται ότι είναι

πάντοτε έτοιμοι να προστεθούν σε καινούργιες συνεδρίες ουσιαστικά μειώνοντας τους γύρους του Sparkle από τρεις σε δύο.

Για να μπορεί ο διαχειριστής να ξεκινάει νέες συνεδρίες, οφείλει να διατηρεί ένα σύνολο *PotentialSigners*. Παρουσιάζουμε δύο διαφορετικές παραλλαγές του συνόλου αυτού, κάθε μια με τα δικά της πλεονεκτήματα και περιορισμούς:

- Παραλλαγή εξίσωσης 11.1: Επιτρέπει σε συμμετέχοντες να συμμετάσχουν σε νέες συνεδρίες αρκεί να μην είναι σε καταστάσεις Blocking2 ή Blocking3 σε κάποια συνεδρία. Αυτή η παραλλαγή επιτυγχάνει σε $f \cdot (n - t + 2) = O(n^2)$ (όπου f είναι το πλήθος των διαθερμαίνων συμμετεχόντων) συνεδρίες αρκεί να υπάρχουν t τίμιοι συμμετέχοντες ανάμεσα στους n.

- Παραλλαγή εξίσωσης 11.2: Σε σχέση με την προηγούμενη εξίσωση *PotentialSigners*, επιπλέον περιορίζει τον αριθμό των συνεδριών στις οποίες ο συμμετέχοντας μπορεί να είναι σε Pending κατάσταση. Έτσι περιορίζει τον κάθε αντίπαλο σε δύο συνεδρίες και συνεπώς το πρωτόκολλο επιτυγχάνει σε $2 \cdot f + 1 = O(n)$ το πολύ συνεδρίες (όπου f είναι το πλήθος των διαθερμαίνων συμμετεχόντων). Το μειονέκτημα είναι ότι προϋποθέτει τουλάχιστον $\frac{f \cdot (t-1)}{2} + t$ έναντι των t της προηγούμενης περίπτωσης.

Για να αποδείξουμε την ορθότητα και την ανεκτικότητα του Sparkling ROAST χρησιμοποιούμε ένα επιχείρημα περιστερώνα (pigeonhole principle) παρόμοιο με αυτό του ROAST [25] αφού πρώτα αποδείξουμε παρόμοιες ιδιότητες με αυτές που παρουσιάζονται στο ROAST.

Ένα ακόμα πλεονέκτημα της κατασκευής μας είναι η επεκτασιμότητά της σε πρωτόκολλα ακόμα περισσότερων γύρων, αρκεί αυτά να έχουν την ιδιότητα της αναγνωρίσιμης αποχώρησης (identifiable abort). Παράδειγμα αποτελεί το πρωτόκολλο Glacius [29] που περιγράφουμε στο προηγούμενο κεφάλαιο.

1.10 Συμπεράσματα και Μελλοντικές Προεκτάσεις

Καθ'όλη τη διάρκεια αυτής της διπλωματικής εργασίας αναλύθηκαν σχήματα πολυυπογραφών και υπογραφών κατωφλίου τύπου Schnorr. Αναλύσαμε επιθέσεις, απειλές, ορισμούς ασφάλειας, συγκρίναμε ιδιότητες και σχήματα ενώ πετύχαμε και ένα νέο ενδιαφέρον ερευνητικό αποτέλεσμα, το πρωτόκολλο Sparkling ROAST.

Ως μελλοντικές προεκτάσεις, οφείλουμε φυσικά να έχουμε κατά νου τις απαιτήσεις του NIST για υπογραφές κατωφλίου [11]. Έτσι λοιπόν παρατηρούμε ότι μελλοντικά σχήματα πρέπει να έχουν (και να συνδυάζουν) αρκετές επιθυμητές ιδιότητες ώστε να υιοθετηθούν σε πρακτικές εφαρμογές (π.χ ευρωστία, προσαρμοστική ασφάλεια, αποδοτικότητα).

Τέλος, διακρίνουμε ένα συγκεκριμένο σημείο που λείπει από τη βιβλιογραφία και αυτό είναι η δημιουργία ενός blind Schnorr threshold σχήματος υπογραφών που να αντέχει στις επιθέσεις ROS.

Chapter 2

Introduction

2.1 Motivation

In the digital age, the integrity and authenticity are foundational pillars of security in al kinds of applications like communication and finance. Cryptographic signatures have long been instrumental in guaranteeing these properties by enabling message authentication and verification in a mathematically rigorous way. Among these, digital signatures based on hard mathematical problems (such as the Discrete Logarithm Problem) have stood the test of time for their simplicity and provable security properties.

Traditional, single-party signature schemes, while effective, are often insufficient in distributed or high-stakes environments where trust cannot be centralized. Multi- and threshold signature schemes address this limitation by distributing signing capabilities among multiple participants. Multi-signatures allow a set of users to jointly produce a single, compact signature while threshold signature schemes enable any subset of size t out of n participants to generate a valid signature, offering fault tolerance and decentralized trust.

Such schemes are especially relevant in real-world applications such as securing blockchain transaction, certificate authorities, distributed key generation protocols and other multi-party computation applications. The practicality of such schemes relies not only on their cryptographic soundness against adversarial behavior in a concurrent setting but also their efficiency and other properties like robustness.

The Schnorr signature scheme with its simplicity and efficiency has emerged as a well-studied foundation for building both multi- and threshold signature schemes. However, as implementations evolve to support multiple signers new vulnerabilities and performance challenges emerge. Notable, attacks such as rogue-key attacks [16], the ROS attack [15] and the Drijvers attack [17] highlight the subtle pitfalls in protocol design, particularly in a concurrent setting where an adversary can open multiple signing sessions at the same time.

2.2 Contributions

This thesis conducts an in-depth study of multi- and threshold Schnorr signature schemes, looking at their security guarantees, performance trade-offs and assumptions

that underlie model signature schemes. We then survey prominent multi- and threshold schemes, examining their algorithmic designs and properties and comparing them to one another. We pay particular attention to the threat landscape, exploring vulnerabilities in schemes that were initially considered secure [16]. Our focus then shifts to the MuSig [18], [19] and FROST [21] , [24] protocol families to look at secure constructions of multi-party signing schemes and look at the trade-offs in communication complexity and round efficiency. We also look at schemes that provide security against adaptive adversaries like the Sparkle+ [28] and Glacius [29] schemes as well as different properties such as robustness [25] .

Building on this groundwork, we introduce Sparkling ROAST, a novel robust wrapper protocol designed to enhance the Sparkle+ threshold signature scheme by adding robustness to the protocol. Two variants of Sparkling ROAST are presented: one prioritizes minimal honest signer assumptions with quadratic overhead in internal sessions, while the other reduces session complexity to linear at the cost of increased honest signer requirements. These constructions exemplify a practical and scalable solution for achieving robust threshold signing in adversarial, decentralized environments.

2.3 Outline of the thesis

This thesis is structured as follows:

- In Chapter 2, we present the necessary cryptographic preliminaries and assumptions.
- In Chapter 3, we introduce the foundational definitions and properties of multi- and threshold signatures.
- In Chapter 4, we look at security threats for multi-party signing schemes looking at how specific attacks were able to break a scheme that was considered secure.
- In Chapter 5, we look at the MuSig schemes (MuSig1 [18] and MuSig2 [19]) discussing how they avoid concurrent attacks and comparing them to each other.
- In Chapter 6, we transition over to the threshold setting by looking at some Distributed Key Generation (DKG) algorithms which are needed for threshold signatures.
- In Chapter 7, we introduce the state of the art FROST schemes [21], [24] and the ROAST protocol which adds robustness to them [25].
- In Chapter 8, we look in much more detail at the unforgeability notions that exist for threshold signatures.
- In Chapter 9, we survey two adaptive secure schemes : Sprakle+ [28] and Glacius [29].

- In Chapter 10, we introduce our new wrapper protocol which adds robustness to the Sparkle+ threshold signing scheme (we call this wrapper protocol Sparkling ROAST).
- In Chapter 11, we conclude the thesis with reflection on our contributions and directions for future research.

Chapter 3

Background

In this chapter, we present some necessary cryptographic tools and definitions which will be used in later sections. We use the notation from [28] throughout this chapter.

3.1 Preliminaries

3.1.1 General Notation

Let $\kappa \in \mathbb{N}$ denote the security parameter and 1^κ be its unary representation.
For a non-empty set S, let $x \xleftarrow{\$} S$ denote sampling an element of S uniformly at random and assigning it to x.
Let $[n]$ denote the set $\{1, \cdots, n\}$ and $[0 \cdots n]$ represent the set $\{0, \cdots, n\}$.
Let *PPT* denote probabilistic polynomial time. Algorithms are randomized unless explicitly noted otherwise. The set of values that have a non-zero probability of being output by an algorithm A on input x is denoted by $[A(x)]$.

Definition 3.1. *Negligible Functions*
A function $negl : \mathbb{N} \to \mathbb{R}$ *is called negligible if for all* $c \in \mathbb{R}, c > 0$, *there exists* $k_0 \in \mathbb{N}$ *such that* $negl(k) < \frac{1}{k^c}$ *for all* $k \in \mathbb{N}, k \geq k_0$.

Group Generation. *Let GrGen be a polynomial-time algorithm that takes as input a security parameter* 1^κ *and outputs a group description* $(\mathbb{G}, p, g)$ *consisting of a group* $\mathbb{G}$ *of order* p, *where* p *is a* κ*-bit prime and a generator* g *of* $\mathbb{G}$.

3.1.2 Hash functions and Idealized Models

Hash Functions. *Hash Functions play fundamental role in modern cryptography. They map elements of a set with a large number of elements to another set with a smaller number of elements. These functions are of the form* $H : X \to Y, |X| > |Y|$ *and* Y *is a finite set. Due to the fact that* X *is a larger set than* Y, *it is obvious that some elements of* X *will be mapped to the same element in* Y. *In general, hash function must have some desirable properties in order to be considered secure. Such properties are:*

- ***Pre-image Resistance:*** *Given a hash value* $y \in Y$ *it is computationally hard to find* $x \in X$ *such that* $H(x) = y$.

- ***Second Pre-image Resistance:*** *Given an element $x_1 \in X$ and its hash value $H(x_1) \in Y$ it is computationally difficult to find element $x_2 \in X$ such that $H(x_1) = H(x_2)$.*

- ***Collision Resistance:*** *It is computationally hard to find two distinct elements $x_1, x_2 \in X$ such that $H(x_1) = H(x_2)$.*

There is a certain hierarchy in those three properties ; namely a hash function that is Collision Resistant is also Second Pre-image Resistant, and a hash function that has the property of Second Pre-image Resistance also has Pre-image Resistance. Equivalently, a hash function that is not Pre-image Resistant cannot be Second Pre-image Resistant and also if it is not Second Pre-image Resistant then it is impossible for it to be Collision Resistant.

From now on, when we refer to hash functions we will assume that they are Collision Resistant and we refer to them as "Cryptographically Secure Hash Function". An example of such function could be one from the SHA family [31].

Random Oracle Model [4]. *The random oracle model (ROM) is an idealized model that treats a hash function H as an oracle in the following way. When queried on an input in the domain of H, the oracle first checks if it has an entry stored in its table for this input. If so, it returns this value. If not, it samples an output in the codomain of H uniformly at random, stores the input-output pair in its table, and returns the output.*

Algebraic Group Model [5]. *The algebraic group model (AGM) is an idealized model that places the following restriction on the adversary. An adversary is algebraic if for every group element $Z \in \mathbb{G} = \langle g \rangle$ that it outputs, it is required to output a representation $\vec{a} = (a_0, a_1, \cdots)$ such that $Z = g^{a_0} \prod Y_i^{a_i}$, where $Y_1, Y_2, \cdots \in \mathbb{G}$ are group elements that the adversary has seen thus far. Intuitively, this captures the notion that an algorithm should know how it computes its outputs from the values it has received as input so far.*

3.1.3 Digital Signatures

Digital signatures are cryptographic mechanisms that provide a way to verify the authenticity, integrity, and origin of a digital message or document. They are the digital equivalent of handwritten signatures or stamped seals, but much more secure and based on mathematical algorithms.

Definition 3.2. *Digital Signatures*
A digital signature scheme consists of the polynomial-time algorithms (Setup, KeyGen, Sign, Verify) defined as follows:

- *Setup$(1^\kappa) \rightarrow par$: On input a security parameter 1^κ, this algorithm outputs public parameters par (which are given implicitly as input to all other algorithms).*

- *KeyGen$() \rightarrow (pk, sk)$: This probabilistic algorithm outputs a public key pk and secret key sk.*

- *$Sign(sk, m) \rightarrow \sigma$: On input a secret key sk and a message m, this algorithm outputs a signature σ.*

- *$Verify(pk, m, \sigma) \rightarrow 0/1$: On input a public key pk, a message m and a signature σ, this deterministic algorithm outputs 1 (accept) if σ is a valid signature on m under pk; ekse, it outputs 0 (reject).*

A digital signature scheme must be correct. It is said to be secure if it is existenstially unforgeable under chosen-message attacks (EUF-CMA).

Definition 3.3. *Correctness*
A digital signature scheme (Setup, KeyGen, Sign, Verify) is correct if for all security parameters $\kappa \in \mathbb{N}$, all key pairs $(pk, sk) \in [KeyGen()]$, and all messages m, we have:

$$\Pr[Verify(pk, m, Sign(sk, m)) = 1] = 1 \tag{3.1}$$

Definition 3.4. *(EUF-CMA)[32]*
We define the EUF-CMA game:

MAIN $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{DS}}(\kappa)$

par $\leftarrow \text{Setup}(1^\kappa)$

$Q_m \leftarrow \emptyset$

$(pk, sk) \leftarrow \text{KeyGen}()$

$(m^*, \sigma^*) \leftarrow \mathcal{A}^{O^{\text{Sign}}}(pk)$

return 0 if $m^* \in Q_m$
$\vee\ \text{Verify}(pk, m^*, \sigma^*) \neq 1$

return 1

$O^{\text{Sign}}(m)$

$Q_m \leftarrow Q_m \cup \{m\}$

$\sigma \leftarrow \text{Sign}(sk, m)$

return σ

Figure 3.1: *The EUF-CMA security game for a digital signature scheme DS = (Setup, KeyGen, Sign, Verify).*

The $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{DS}}$ gives to a PPT adversary $\mathcal{A}$ access to a signing oracle $O^{\text{Sign}}(m)$ allowing him to get polynomial-many signatures of the (unknown to him) secret key sk. The adversary wins the game if he is able to produce a signature on any message m^* for which he has not queried the signing oracle.

Let the advantage of an adversary $\mathcal{A}$ playing the EUF-CMA game $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{DS}}$ as defined in Figure 3.1, be as follows:

$$\text{Adv}^{\text{EUF-CMA}}_{\mathcal{A},\text{DS}}(\kappa) = \left|\Pr\left[\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{DS}}(\kappa) = 1\right]\right| \tag{3.2}$$

A digital singature scheme DS = (Setup, KeyGen, Sign, Verify) is existentially unforgeable under chosen-message attacks if for all PPT adversaries $\mathcal{A}$, $\text{Adv}^{\text{EUF-CMA}}_{\mathcal{A},\text{DS}}(\kappa)$ is negigible.

3.1.4 Schnorr Identification Protocol

Let $\mathbb{G}$ be a cyclic group of prime order p with generator g.
The Schnorr Identification Protocol [7] is a Zero-Knowledge protocol involving a Prover and a Verifier. In this protocol the Prover proves to the Verifier that he knows the discrete logarithm x of a group element $X \in \mathbb{G}$ without revealing any information on the value of x.
The steps of the protocol are the following:

1. ***Commitment phase*** *: The Prover samples a random value $r \xleftarrow{\$} \mathbb{Z}_p$ and calculates his commitment $R = g^r$ which he sends to the verifier.*
2. ***Challenge phase*** *: The Verifier samples a random value $c \xleftarrow{\$} \mathbb{Z}_p$ which he sends to the Prover.*
3. ***Response phase*** *: The Prover calculates his response to the challenge c, $z = r + c \cdot x$.*
4. ***Verification phase*** *: After the Verifier receives the response z, he checks if $g^z \stackrel{?}{=} R \cdot X^c$. If the equation holds, then he accepts ; otherwise rejects.*

Intuitively, the Schnorr Identification Protocol has zero-knowledge since the response z is a line equation with gradient the challenge. However, since the verifier does not know the value r , it is impossible for him to find x. The security of this protocol relies on the Discrete Logarithm Assumption which we discuss later in this chapter.

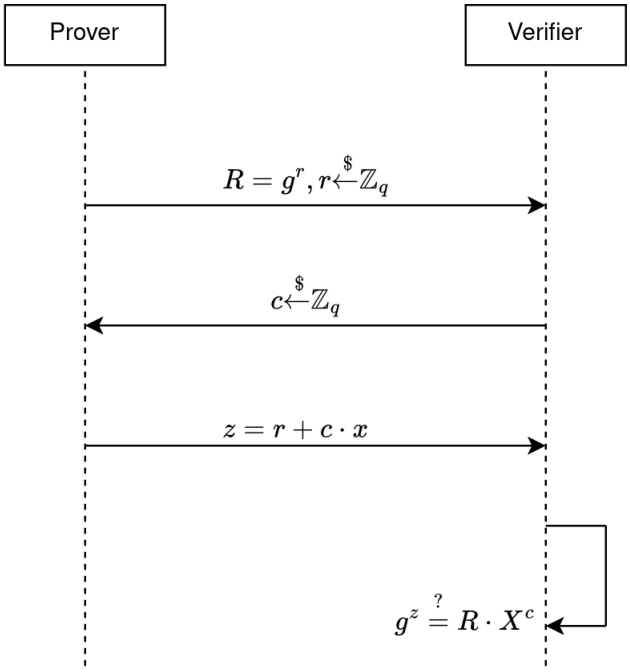

Figure 3.2: *The Schnorr Identification Protocol.*

3.1.5 The Fiat-Shamir Transformation

The Fiat-Shamir transformation [8] turns an interactive $\sum$-protocol into a non-interactive proof system by replacing the verifier's random choice of c with $c = H(\{P\})$ where H is a

hash function and $\{P\}$ is the set of public information of the protocol which the verifier has access to. Hash functions are treated like Random Oracles (we are working in the ROM) and therefore we consider the output of a hash function to be random. By doing this, we can turn interactive proof systems non-interactive ; these new proof systems can be used as signatures. An example use case of the Fiat-Shamir transfomation is turning the Schnorr Identification protocol into a digital signature scheme, the very well-known and studied Schnorr signature which we describe next.

3.1.6 Schnorr Signatures

Definition 3.5. *Schnorr Signatures [6]*
The Schnorr signature scheme consists of polynomial-time algorithms (Setup, KeyGen, Sign, Verify), defined as follows:

- *$Setup(1^\kappa) \rightarrow par$: On input a security parameter 1^κ, run $(\mathbb{G}, p, q, g) \leftarrow GrGen(1^\kappa)$ and select a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$. Output public parameters $par \leftarrow ((\mathbb{G}, p, g), H)$ (which are given implicitly as input to all other algorithms).*
- *$KeyGen() \rightarrow (pk, sk)$: Sample a secret key $x \xleftarrow{\$} \mathbb{Z}_p$ and compute the public key as $X \leftarrow g^x$. Output key pair $(pk, sk) \leftarrow (X, x)$.*
- *$Sign(sk, m) \rightarrow \sigma$: On input secret key $sk = x$ and a message m, sample a nonce $r \xleftarrow{\$} \mathbb{Z}_p$. Then, compute a nonce commitment $R \leftarrow g^r$, the challenge $c \leftarrow H(X, m, R)$ and the response $z \leftarrow r + c \cdot x$. Output a signature $\sigma \leftarrow (R, z)$.*
- *$Verify(pk, m, \sigma) \rightarrow 0/1$: On input a public key $pk = X$, a message m and a signature $\sigma = (R, z)$, compute $c \leftarrow H(X, m, R)$ and output 1 (accept) if $g^z = R \cdot X^c$; else, output 0 (reject).*

It is clear that the Schnorr signature scheme is derived from the Schnorr Identification protocol with the use of the Fiat-Shamir transform. In more detail, instead of having a verifier provide a challenge c to the Prover, we create the challenge c by hashing all the public information of the protocol (that is the public key X of the signer, the message m to be signed and the commitment of the protocol R). This way, any public verifier who has access to the message m and the signature $\sigma = (R, z)$ can check the validity of the signature. The Schnorr signature scheme is secure under tthe discrete logarithm assumption in the ROM (Random Oracle Model) [33].

3.1.7 Polynomial Interpolation

A polynomial $f(x) = a_0 + a_1 \cdot x + \cdots a_t \cdot x^t$ of degree t over a field $\mathbb{F}$ can be interpolated by $t+1$ points. Let $S \subset [n]$ be the list of $t+1$ distinct indices corresponding to the x-coordinates $x_i \in \mathbb{F}, i \in S$ of these points. Then the Lagrange polynomial $L_i(x)$ has the form:

$$L_i(x) = \prod_{j \in S j \neq i} \frac{x - x_j}{x_i - x_j} \tag{3.3}$$

Given a set of $t+1$ points $(x_i, f(x_i))_{i\in[t+1]}$, any point $f(x_\ell)$ on the polynomial f can be determined by Lagrange interpolation as follows:

$$f(x_\ell) = \sum_{k\in S} f(x_k) \cdot L_k(x_\ell) \tag{3.4}$$

Definition 3.6. *Shamir Secret Sharing [1]*
Shamir secret sharing is an (n, t)-threshold secret sharing scheme consisting of algorithms (IssueShares, Recover), defined as follows:

- *IssueShares$(x, n, t) \to \{(1, x_1), \cdots, (n, x_n)\}$: On input a secret x, number of participants n and threshold t, perform the following. First, define a polynomial $f(Z) = x + a_1 \cdot Z + a_2 \cdot Z^2 + \cdots + a_{t-1} \cdot Z^{t-1}$ by sampling $t-1$ coefficients at random: $a_1, \cdots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$. Then, set each participant's share $x_i, i \in [n]$, to be the evaluation of $f(i)$:*

$$x_i = x + \sum_{j\in[t-1]} a_j \cdot i^j \tag{3.5}$$

  Output $\{(i, x_i)\}_{i\in[n]}$.

- *Recover$(t, \{i, x_i\}_{i\in S}) \to \perp / x$: On input threshold t and a set of shares $\{(i, x_i)\}_{i\in S}$, output $\perp$ if $S \not\subseteq [n]$ or if $|S| < t$. Otherwise, recover x as follows:*

$$x \leftarrow \sum_{i\in S} \lambda_i \cdot x_i \tag{3.6}$$

  where the Lagrange coefficient for the set S is defined by

$$\lambda_i = \prod_{j\in S, j\neq i} \frac{j}{i-j} \tag{3.7}$$

3.1.8 Additive Secret Sharing

While Shamir secret sharing and derived constructions require shares to be points on a secret polynomial f where $f(0) = s$, an additive secret sharing scheme allows a set of a participants to jointly compute a shared secret s by each participant P_i contributing a value s_i such that the resulting shared secret $s = \sum_{i\in[a]} s_i$, the summation of each participant's share.

Additive Secret Sharing is very useful when a set of participants want to create a joint secret s and do not trust a single party to create that secret.

3.2 Assumptions

3.2.1 Discrete Logarithm Assumption (DLOG)

Definition 3.7. *Discrete Logarithm Problem (DLP)*

Let $\mathbb{G}$ be a finite group of order p and g one of its generators. Given $h \in \mathbb{G}$ find $x \in \mathbb{Z}_p$ such that $g^x = h$.

Definition 3.8. *Discrete Logarithm Assumption (DLOG)*
Let the advantage of an adversary $\mathcal{A}$ playing the discrete logarithm game, $Game^{dl}$ as defined in Figure 3.3 be as follows:

$$\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{A}}(\kappa) = \left|\Pr\left[\mathsf{Game}^{\mathsf{dl}}_{\mathcal{A}}(\kappa) = 1\right]\right| \tag{3.8}$$

The discrete logarithm assumption holds if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{A}}(\kappa)$ is negligible.

MAIN $\mathrm{Game}^{\mathsf{dl}}_{\mathcal{A}}(\kappa)$

$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

$x \leftarrow_{\$} \mathbb{Z}_p; \quad X \leftarrow g^x$

$x' \leftarrow \mathcal{A}((\mathbb{G}, p, g), X)$

if $x' = x$

return 1

return 0

Figure 3.3: *The discrete logarithm (DL) game for adversary $\mathcal{A}$.*

3.2.2 Algrebraic One-More Discrete Logarithm Assumption (AOMDL)

Definition 3.9. *(Algebraic) One-More Discrete Logarithm Problem [19]*
Let $\mathbb{G}$ be a finite group of order p and g one of its generators. Given $t+1$ values $X_0, X_1, \cdots, X_t$ and the ability to make t queries to a discrete logarithm oracle O^{dl} find $x_0, x_1, \cdots, x_t$ such that $g^{x_i} = Xi, i \in [0, \cdots, t]$.

Definition 3.10. *Alegbraic One-More Discrete Logarithm Assumption*
Let the advantage of an adversary $\mathcal{A}$ playing the $t+1$-algebraic one-more discrete game, $Game^{t+1\text{-}aomdl}_{\mathcal{A}}(\kappa)$, as defined in Figure 3.4, be as follows:

$$\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{A}}(\kappa) = \left|\Pr\left[\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{A}}(\kappa) = 1\right]\right| \tag{3.9}$$

The algebraic one-more discrete logarithm assumption holds if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{A}}(\kappa)$ is negligible.

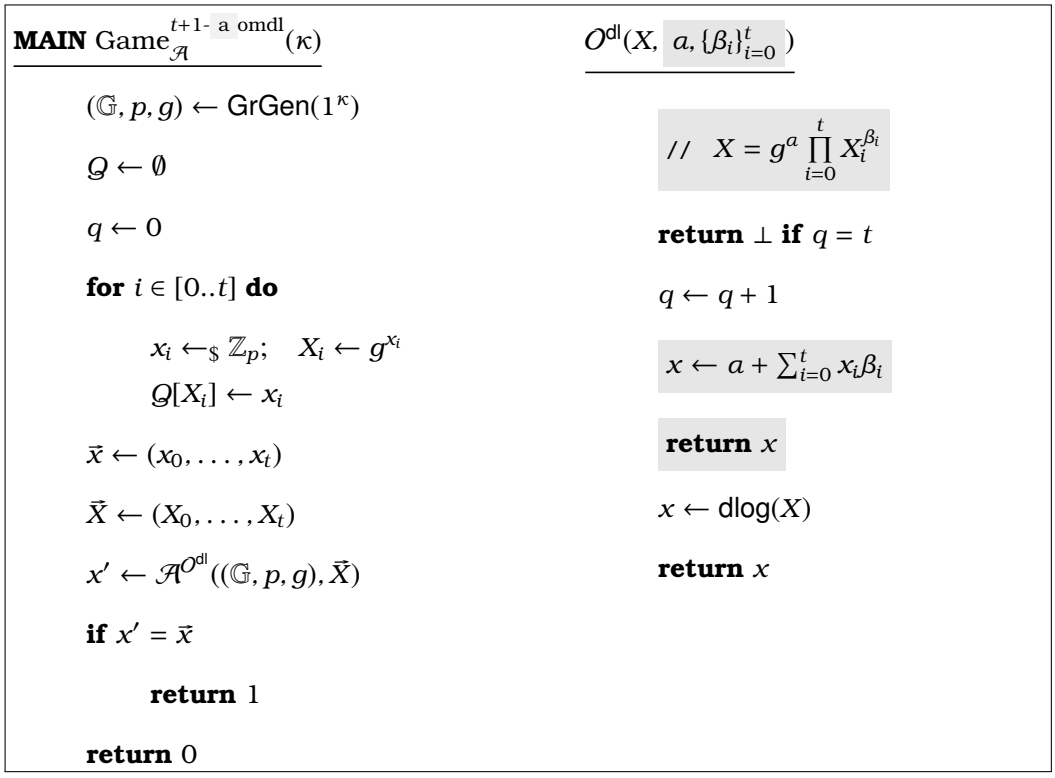

Figure 3.4: *The Algebraic One-More Discrete Logarithm (AOMDL) game. The difference between the OMDL and AOMDL games are highlighted in gray. The key distinction is that in the AOMDL game, the adversary can only query its discrete logarithm oracle on linear combinations of its challenge group elements whereas in the OMDL game, the environment must return the discrete logarithm of an arbitrary group element.* dlog *is an algorithm that finds the discrete logarithm of an arbitrary group element X base g.*

Chapter 4

Multi- and Threshold Signatures

In this chapter, we introduce the notion of multi-party signatures as well as threshold signatures. We describe the algorithms and properties they should hold. We also present some applications for such schemes. Due to the focus of this thesis being Schnorr signatures ; we discuss their advantages over other multi-party signature schemes. We follow the notation of [29] for defining multi- and threshold signatures as well as the game definitions for Correctness and Unforgeability properties.

4.1 Multi- and Threshold Signature Schemes

4.1.1 Multi-signatures

Multi-signatures protocols, first introduced in [34] allow a group of signers (each possessing its own private/ public key pair) to jointly produce a single signature σ on a message m. A trivial way to transform a standard signature scheme into a multi-signature scheme is to have each signer produce a stand-alone signature for m with its private key and to concatenate all individual signatures. However, the size of the multi-signature in that case grows linearly with the number of signers. In order to be useful and practical, a multi-signature scheme should produce signatures whose size is (ideally) independent from the number of signers and close to the one of an ordinary signature scheme.

4.1.2 Threshold signatures

Threshold signature protocols are quite similar to multi-signatures in the sense that multiple parties are needed in order for a signature to be produced. The main difference between threshold signature protocols and multi-signatures is that in the former a threshold t out of n parties are needed for a signature to be produced whereas in the latter all n out of n parties need to participate in the signing session. Same as before, each party holds a private/public key pair and the final group signature should be independent from the number of signers.

4.1.3 Coordinator Role

In both types of schemes, we could consider that a coordinator is present whose job is to propagate messages between signers and report misbehaving participants. Although this

coordinator is not needed in all schemes, his presence can reduce communication costs. We note that the coordinator is semi-trusted meaning that he cannot break the unforgeability of said schemes but can mount denial-of-service attacks. Throughout this thesis, we will assume that a coordinator is present for every protocol.

4.1.4 Authenticated Channels

Throughout this entire thesis we consider that each protocol message exchanged between signing parties is sent via an authenticated channel, meaning that each protocol message is accompanied with a signature of the sender using a separate single-party Digital Signing scheme DS. In our notation we skip the signatures produced by DS for clarity however each time a signer receives a protocol message, he must check the validity of the DS signature.

4.2 Definition of the algorithms

Definition 4.11. *Multi- and Threshold Signatures*
A multi-signature scheme M *or a threshold signature scheme* TS *consisting of* r *signing rounds and a signing set SS (*$SS = [n]$ *in the case of multi-signatures and* $SS \subseteq [n]$ *with* $SS \geq t$ *in the case of threshold signatures) is a tuple of algorithms* $\mathsf{M} = (Setup, KeyGen, (Sign_1, Sign_2, \cdots, Sign_r), Combine, Verify)$ *or* $\mathsf{TS} = (Setup, KeyGen, (Sign_1, Sign_2, \cdots, Sign_r), Combine, Verify)$ *defined as follows:*

- $Setup(1^\kappa, n, t) \rightarrow par$*: On input a security parameter* 1^κ*, the number* n *of total signers and a threshold* t *(*$t = n$ *in the case of multi-signatures and* $t < n$ *for threshold signatures) run* $par \leftarrow GrGen(1^\kappa)$*. Output public parameters par (which are given implicitly as input to all other algorithms).*

- $KeyGen(n) \rightarrow (pk, \{(pk_i, sk_i)\}_{i\in[n]})$*: A probabilistic algorithm that takes an input the number of signers* n *and outputs the public key* pk *representing the set of signers (group's public key), the set* $\{pk_i\}_{i\in[n]}$ *of public keys for each signer, and the set* $\{sk_i\}_{i\in[n]}$ *of secret keys for each signer. It it assumed that participant* P_i *is sent its secret key* sk_i *privately.*

- $Sign = (Sign_1, Sign_2, \cdots, Sign_r)$ *: The signing protocol is split into* r *algorithms:*

  - $Sign_k(SS, m, i, (pm_{k-1,j})_{j\in SS}, sk_i, st_i) \rightarrow (pm_{k,i}, st_i)$ *: The* k*-th round signing algorithm for* $k \in [r]$ *which takes as input the signer set SS consisting of the signing parties of the session, a message* m*, an index* $i \in [n]$*, a tuple of protocol messages of the* $(k-1)$*-th round* $(pm_{k-1,j})_{j\in SS}$*, a secret signing key* sk_i *and a state* st_i*. It outputs a protocol message* $pm_{k,i}$ *for the* k*-th round and the updated state* st_i*. We define* $pm_{0,j} \leftarrow \perp$ *for all* $j \in SS$*.*

- $Combine(SS, m, (pm_{k,i})_{k\in[r],i\in SS}) \rightarrow \sigma$*: The deterministic combine algorithm which takes as input the signer set SS, a message* m *and a tuple of protocol messages* $(pm_{k,i})_{k\in[r],i\in SS}$ *and outputs a signature* σ*.*

- *Verify*$(pk, m, \sigma) \to 0/1$: *The deterministic verification algorithm takes as input a public key* pk*, a message* m *and a signature* σ *and outputs either* 1 *(accept) or* 0 *(reject).*

4.3 Properties of multi- and threshold signature schemes

4.3.1 Correctness

Definition 4.12. *Correctness of Multi- and Threshold Signature schemes*
Consider the game $\textbf{Game}^{\text{cor}}_{\text{M,TS}}$ *defined in Figure 4.1. Then an* r*-round multi-signature scheme* M *or an* r*-round threshold signature scheme* TS *is correct, if for all* $\kappa \in \mathbb{N}$*,* n, t *with* $t \leq n$*, messages* $m \in \mathcal{M}$ *(where* $\mathcal{M}$ *is the message space),* $SS \subseteq [n]$*, the following holds:*

$$\Pr\left[\mathit{Game}^{\text{cor}}_{\text{M,TS}}(1^{\lambda}, n, t, \text{SS}, m) \Rightarrow 1\right] \geq 1 - \text{negl}(\lambda). \tag{4.1}$$

$\textbf{Game}^{\text{cor}}_{\text{M,TS}}(1^{\lambda}, n, t, \text{SS}, m)$

for $i \in \text{SS} : \text{st}_i := \emptyset$

$\text{par} \leftarrow \text{Setup}(1^{\kappa}, n, t)$

$(\text{pk}, \{\text{pk}_i, \text{sk}_i\}_{i\in[n]}) \leftarrow \text{KeyGen}(\text{par})$

for $i \in \text{SS} : \text{pm}_{0,i} := \perp$

for $k \in [r]$:

  for $i \in \text{SS}$:

    $(\text{pm}_{k,i}, \text{st}_i) \leftarrow \text{Sign}_k(\text{SS}, m, i, (\text{pm}_{k-1,j})_{j\in\text{SS}}, \text{sk}_i, \text{st}_i)$

$\sigma := \text{Combine}(\text{SS}, m, (\text{pm}_{k,i})_{k\in[R],i\in\text{SS}})$

return $\text{Verify}(\text{pk}, m, \sigma)$

Figure 4.1: *Correctness game for an* r*-round multi- or threshold signature scheme*

4.3.2 Unforgeability

We provide a basic game definition for unforgeability in Figure 4.2 (we discuss unforgeability and security models of threshold signatures extensively in later chapters). Let $\mathcal{A}$ *be the adversary in this game. Initially,* A *gets the public parameters* par*, an honestly generated public key* pk *and threshold public keys* $\{pk_i\}_{i\in[n]}$ *of all signers as input. At the beginning of the game,* $\mathcal{A}$ *must provide the set* C *of parties he wishes to corrupt.* $\mathcal{A}$ *learns the secret key* sk_i*,* $i \in C$ *and has access to their internal state* st_i*. At any point in time* $\mathcal{A}$ *can start a new signing session with identifier* sid *for signer set* SS *and message* m *by calling the oracle* $NEXT(sid, SS, m)$*. We allow* $\mathcal{A}$ *to participate in any number of concurrent signing sessions. Further,* $\mathcal{A}$ *can interact with honest signers using the signing oracles* SIG_k *for*

$k \in [r]$. $\mathcal{A}$ *can query each of these oracles for an individual honest signer* i *and a session identifier* sid. *When querying* SIG_k, $\mathcal{A}$ *can freely choose the protocol messages* pm_{k-1} *of the* $(k-1)$*-th round. Importantly, we do not assume broadcast channels for the signing protocol, and the adversary could send different messages to different honest signers. However, we do assume authenticated channels, and our unforgeability game captures this via the Allowed algorithm. The Allowed algorithm also enforces that* $\mathcal{A}$*'s queries for each session are consistent. Finally, when* $\mathcal{A}$ *outputs a forgery* (m^*, σ^*)*, we say that* $\mathcal{A}$ *wins if he has not initiated a signing session for the message* m^*.

Definition 4.13. *Unforgeability of Multi- and Threshold Signature schemes*
Let M *be an* r*-round multi-signature scheme or* TS *be an* r*-round threshold signature scheme. Consider* ***Game*** $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{M,TS}}(1^\kappa, n, t)$ *in Figure 4.2. We say that a scheme is secure if for all* $\kappa \in \mathbb{N}$, n, t *with* $t \leq n$, *PPT adversaries* $\mathcal{A}$, *the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\mathsf{M,TS}}(1^\kappa, n, t) := \Pr\left[\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS,M}}(1^\kappa, n, t) \Rightarrow 1\right]. \tag{4.2}$$

In summary, the adversary should not be able to produce a forgery even if he controls $n-1$ *signers of a multi-signature scheme or* $t-1$ *of a threshold signing scheme.*

4.4 Multi- and Threshold Schnorr Signatures

In this thesis, we mainly focus on Schnorr signatures both in the multi-signing as well as the threshold setting. The main characteristic of these schemes is that the output signature of the Combine algorithm looks exactly the same as a single-party Schnorr signature (as discussed in Section 3.1.6). This means that a single-party Schnorr verifier can verify the validity of the multi-party signature without changing his Verify algorithm. This provides backwards compatibility with older systems as multi- and threshold signing algorithms are used for a constantly growing number of applications. Another important thing to note is that the verifier cannot distinguish between a single-party Schnorr signature or a Schnorr signature produced via a multi- or threshold signing scheme.

4.5 Applications of Multi- and Threshold Signatures

In the last few years, there has been a large increase in the amount of research for multi- and threshold signatures. This is mainly because getting a signature from a number of participants of a signing set is much more secure than getting a single-party signature where the signer could have been corrupted. However, another reason is that large blockchains like Bitcoin [9] have adopted Schnorr signatures after using ECDSA [35] for a long time [36]. The fact that NIST has made calls for threshold schemes [11],[10] is a clear indication that the use of multi- and threshold signature schemes will be greatly increased in the years to come. Some real-world applications where such schemes can be used include:

- *Securing bitcoin wallets: The very nature of a blockchain system is decentralized and the key idea is the distribution of trust among the participants. A single signing*

Game $\text{UF-CMA}^{\mathcal{A}}_{\text{M,TS}}(1^\kappa, n, t)$

par $\leftarrow$ Setup$(1^\kappa, n, t)$

$(\text{pk}, \{\text{pk}_i, \text{sk}_i\}_{i\in[n]}) \leftarrow$ KeyGen(par)

$C := \{c_1, \cdots, c_{t-1}\},\ \mathcal{H} := [n] - C$

if $C \geq t$: **return** $\perp$

for $i \in C$:$st_{\mathcal{A}} = st_{\mathcal{A}} \bigcup (sk_i, st_i)$

pmsg := $\emptyset$, Queried := $\emptyset$

SIGN := (NEXT, $(\text{SIGN}_k)_{k\in[r]}$)

$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\text{Next,Sign}_k}(\text{pk}, \{\text{pk}_i\}_{i\in[n]})$

if $m^* \in$ Queried: **return** 0

return Ver(pk, m^*, σ^*)

Oracle NEXT(sid, SS, m)

if $(|\text{SS}| < t) \vee (\text{SS} \nsubseteq [n])$: **return** $\perp$

if sid $\in$ Sessions: **return** $\perp$

Sessions := Sessions $\cup$ {sid}

Queried := Queried $\cup$ $\{m\}$

message[sid] := m

signers[sid] := SS

for $i \in$ SS: round[sid, i] := 1

Oracle Sign_k(sid, SS, m, i, $(\text{pm}_{k-1,j})_{j\in\text{SS}}$)

input := (SS, m, i, $(\text{pm}_{k-1,j})_{j\in\text{SS}}$)

if Allowed(sid, k, input) = 0:
return $\perp$

$(\text{pm}_{k,i}, \text{st}_i) \leftarrow \text{Sign}_k(\text{input}, \text{sk}_i, \text{st}_i)$

pmsg[sid, k, i] := $\text{pm}_{k,i}$

round[sid, i] := $k + 1$

return $\text{pm}_{k,i}$

Allowed(sid, k, SS, m, i, $(\text{pm}_{k-1,j})_{j\in\text{SS}}$)

assert sid $\in$ Sessions

assert SS = signers[sid]

assert i $\in (SS \cap \mathcal{H})$

assert k = round[sid, i]

assert m = message[sid]

if $k = 0$: **return** 1

for $j \in$ SS $\setminus$ $\{i\}$:

if
$\text{pm}_{k-1,j} \notin$ pmsg[sid, $k-1, j$]:
return 0

return 1

Figure 4.2: *UF-CMA game for an r-round multi- or threshold signature scheme*

identity cancels the notion of distributed trust and introduces a single point of failure. To protect against theft, a user distributes t-out-of-n shares of her private key among n devices that she owns. When she initiates a Bitcoin transaction from one device, the transaction would have to be approved by $t-1$ other devices in order to reach the threshold. An attacker will have to compromise t of the user's devices to steal her bitcoins. The same concept can be used for wallet recovery[37], [38]. A real world example is Frostsnap [39].

- *Certification Authority and Directory Service : A certification authority (CA) is a service run by a trusted organization that verifies and confirms the validity of public keys. The issued certificates usually also confirm that the real-world user defined in a certificate is in control of the corresponding private key. A certificate is simply a digital signature under the CA's private signing key on the public key and the identity (ID) claimed by the user. A secure directory service maintains a database of entries, processes lookup queries, and returns the answers authenticated by a signature under its private signing key. DNS authentication is one example of a service that can be distributed [40], [41], [42].*

- *Distributed random beacons: Active adversaries can behave dishonestly in order to bias public random choices toward their advantage. public randomness are manifold and include the protection of hidden services in the Tor network ,selection of elliptic curve parameters , Byzantine consensus , electronic voting and much more. Distributed protocols can be used in order to generate bias-resistant, third-party verifiable randomness. Threshold signatures can be used to verify the distributed nature of the output [43].*

- *E-voting and whistleblowing via threshold ring signatures: Suppose that a member of a national parliament (an MP) would like to submit a controversial bill for a law. The bill is controversial enough that the MP could lose his standing among his own party. However, if enough other members agree to the bill, it will be submitted for an official law. The solution, then, is for the first MP to publish their bill using a threshold ring signature with strong inter-signer anonymity. Any other MP can add themselves by contributing to the signature [44].*

We will discuss further application of each scheme we present in further chapters.

4.6 Metrics of Complexity in Multi- and Threshold signatures

In order to study the complexity of different schemes and compare them to each other, we will use the metrics defined in [12]. These are:

- ***Rounds**: The number of communication rounds between the signers*

- ***Communication**: The amount of bits exchanged during the protocol, the elements exchanged can have the following sizes:*

- $\mathbb{Z}_p$: *A scalar.*
- $\mathbb{G}$: *A group element.*

- ***Computation**: The amount of operations each signer has to make, these are:*
  - *GMul: Multiplication of group elements of size $\mathbb{G}$.*
  - *GExp: Raising a group element of size $\mathbb{G}$ to a scalar element $\mathbb{Z}_p$.*
  - *SMul: Multiplication of scalars of size $\mathbb{Z}_p$ each.*
  - *Lagr: Calculation of a Lagrange coefficient (this can actually be calculated as $t \cdot SMul + t \cdot (SInv + SMul)$ where t is the size of the signing set and SInv is the cost of calculating the inverse of a scalar, but we include it for simplicity).*
  - *Hh: Hashing, we consider all hashes to have the same cost.*

Chapter 5

Security Concerns of Multi- and Threshold signatures

5.1 The ROS problem

The ROS problem (Random inhomogeneities in a Overdetermined Solvable system of linear equations) first introduced in [13] is a well-known problem in the field of blind signatures ; mainly blind constructions of Schnorr signatures such as [45] and Okamoto-Schnorr blind signatures [46]. In his 2001 paper, Schnorr proved that this problem plays a pivotal role in assessing the security of such schemes as finding a solution to the problem would allow an adversary to mount an attack on said schemes breaking the "one-more" unforgeability aspect that blind signatures require. In more detail, it would allow an adversary to forge $\ell + 1$ signatures after interacting only ℓ times with a legitimate signer. We next describe the ROS problem:

Definition 5.14. *ROS problem*
Given a prime number p and access to a random oracle H_{ros} with range in $\mathbb{Z}_p$, the ROS problem (in dimension ℓ) asks to find $\ell + 1$ vectors $\vec{p}_i \in \mathbb{Z}_p^\ell$ and a vector $\vec{c} = (c_1, \cdots, c_\ell)$ such that:

$$H_{ros}(\vec{p}_i) = \langle \vec{p}_i, \vec{c} \rangle \quad \text{for all } i \in [\ell + 1]. \tag{5.1}$$

Equivalently,

$$\begin{pmatrix} H_{ros}(\vec{p}_1) \\ H_{ros}(\vec{p}_2) \\ \vdots \\ H_{ros}(\vec{p}_\ell) \\ H_{ros}(\vec{p}_{\ell+1}) \end{pmatrix}_{(\ell+1)\times 1} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,\ell} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,\ell} \\ \vdots & \vdots & \ddots & \vdots \\ p_{\ell,1} & p_{\ell,2} & \cdots & p_{\ell,\ell} \\ p_{\ell+1,1} & p_{\ell+1,2} & \cdots & p_{\ell+1,\ell} \end{pmatrix}_{(\ell+1)\times \ell} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{pmatrix}_{\ell \times 1} \tag{5.2}$$

When first looking at this problem one might wonder why would we care about it in the setting of multi- and threshold signatures. However, as it will become clear by the end of this chapter, blind signatures and threshold signatures are quite similar in the sense that there is interaction between the signers just like there is interaction between the User and Signer in blind signature schemes. In blind signatures, an adversarial User will try to

get forgeries against the Signer ; whereas in multi- and threshold signatures adversarial signers will try to forge signatures on behalf of the group.

5.2 Wagner's Generalized Birthday Attack

In a very well known paper [14], Wagner studied a generalization of the birthday attack which he called the k-sum problem.
The birthday problem is a very famous and well-studied combinatorial tool in cryptography which states the following:

Definition 5.15. *Birthday Problem*
Given two lists L_1, L_2 of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $x_1 \in L_1$ and $x_2 \in L_2$ such that $x_1 \oplus x_2$.

This problem has been studied extensively and it can be proven that a solution x_1, x_2 exists with good probability once $|L_1| \times |L_2| \gg 2^n$ holds. The complexity of the optimal algorithm is $O(2^{n/2})$. A solution to this problem can be found by using the join operator $L_1 \bowtie L_2$. The cost of computing a join between the two lists is $|L_1| + |L_2|$ steps of computation.

5.2.1 Generalized Birthday Problem / k-sum problem

Definition 5.16. *Generalized Birthday Problem/k-sum problem*
Given k lists $L_1, \cdots, L_k$ of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $x_1 \in L_1, x_2 \in L_2, \cdots, x_k \in L_k$ such that $x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$.

5.2.2 Wagner's Algorithm

Wagner studied the k-sum problem and came up with an algorithm to solve it. In order to understand the generalized algorithm for k lists let's study the case where $k = 4$. We are given 4 lists and our goal is to find $x_1 \in L_1, x_2 \in L_2, x_3 \in L_3, x_4 \in L_4$ that XOR to zero. Let $low_l(x)$ be a function that denotes the ℓ least significant bits of x. Based on this function we can define the generalized join operator $\bowtie_\ell$ so that $L_1 \bowtie_\ell L_2$ contains all pairs in $L_1 \times L_2$ that agree in their ℓ least significant bits. Given this new operator we find some very important properties:

1. $low_\ell(x_i \oplus x_j) = 0 \iff low_\ell(x_i) = low_\ell(x_j)$
2. *If $x_1 \oplus x_2 = x_3 \oplus x_4$, then $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$*
3. *If $low_\ell(x_1 \oplus x_2) = 0$ and $low_\ell(x_3 \oplus x_4) = 0$ then we have $low_l(x_1 \oplus x_2 \oplus x_3 \oplus x_4) = 0$ and in this case $Pr[x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0] = 2^\ell / 2^n$*

With these properties in mind, we can follow these steps in order to solve the 4-sum problem:

ALGORITHM 5.1: *List Matching Based on Property 2*

1: Extend the lists $L_1, \ldots, L_4$ until they each contain about 2^ℓ elements.
2: Generate list L_{12} of values $x_1 \oplus x_2$ such that $\text{low}_\ell(x_1 \oplus x_2) = 0$.
3: Generate list L_{34} of values $x_3 \oplus x_4$ such that $\text{low}_\ell(x_3 \oplus x_4) = 0$.
4: Search for matches between L_{12} and L_{34}. Any match will satisfy $x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

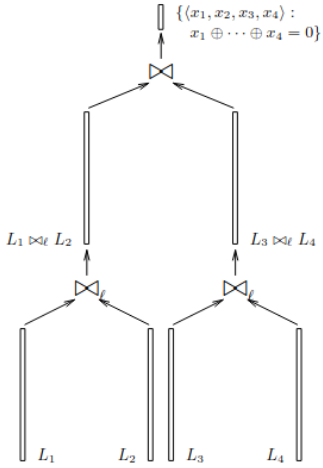

Figure 5.1: *Illustration of the 4-sum algorithm steps*

We can calculate the amount of steps needed for this algorithm to conclude by figuring out how large ℓ must be.
We can calculate the expected number of elements in lists L_{12} and L_{34} after the joins $\bowtie_\ell$:

$\mathbb{E}[|L_{12}|] = |L_1| \times |L_2|/2^\ell = 2^{2\ell}/2^\ell = 2^\ell$
$\mathbb{E}[|L_{34}|] = |L_3| \times |L_4|/2^\ell = 2^{2\ell}/2^\ell = 2^\ell$

By property 3, any pair in $L_{12} \times L_{34}$ yields a match with probability $2^\ell/2^n$. Therefore, the expected number of matching elements between L_{12} and L_{34} is $|L_{12}| \times |L_{34}|/2^{n-\ell}$. This number is at least 1 when $\ell \geq n/3$. Therefore the resulting algorithm can be implemented with $O(2^{n/3})$ time and space.

Note : We can easily extend this algorithm to find elements that XOR to a different value a i.e $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = a$ by simply changing the lists $L_2' = L_2 \oplus a$ and $L_4' = L_4 \oplus a$ and computing : $(L_1 \bowtie_\ell L_2') \bowtie (L_3 \bowtie_\ell L_4')$

Generalizing for all k : In [14] it is shown how to generalize this algorithm for k lists. In short, in that case we will have a tree like the one in Figure 5.1 but of depth $\log(k)$. It is proven that that algorithm requires $O(k \cdot 2^{n/(1+\log(k))})$ time and space and uses lists of size $O(2^{n/(1+\log(k))})$.

5.2.3 Using Wagner's Algorithm against ROS

Wagner's k-sum algorithm can be extended to use other operators apart from $\oplus$. By using addition, we can solve instances of the ROS problem. We fist notice that is easy to find ℓ

partial solution to the ROS problem like so:

$$\begin{pmatrix} H_{ros}(\vec{p}_1) \\ H_{ros}(\vec{p}_2) \\ \vdots \\ H_{ros}(\vec{p}_\ell) \\ H_{ros}(\vec{p}_{\ell+1}) \end{pmatrix}_{(\ell+1)\times 1} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ ? & ? & \cdots & ? \end{pmatrix}_{(\ell+1)\times \ell} \cdot \begin{pmatrix} H_{ros}(\vec{p}_1) \\ H_{ros}(\vec{p}_2) \\ \vdots \\ H_{ros}(\vec{p}_\ell) \end{pmatrix}_{\ell\times 1} \tag{5.3}$$

Notice that the first ℓ lines of the equation 5.3 hold trivially. The computationally hard problem is to find the last partial solution, that is, a non-trivial linear combination $\vec{p}_{\ell+1}$ of these values c_i that matches the hash image of $H_{ros}(\vec{p}_{\ell+1})$. Also, notice that the trivial solutions do not have to be unit vectors, we can use any constant value a in our non-zero vector coefficients as long as the matching c_i is $H_{ros}(\vec{p}_i)/a$.

Solving the ROS problem is surprisingly straight-forward using Wagner's algorithm. We start of by fixing the last line of the matrix to $\vec{p}_{\ell+1} = [1, 1, \cdots, 1]$. We populate lists $L_1, \cdots, L_\ell$ with hash values and then use the k-sum algorithm to find a hash value from each list such that they sum up to $H_{ros}(\vec{p}_{\ell+1})$.

We discuss how to populate list L_1 and the rest will be the same. The elements in L_1 correspond to those with the first line of the matrix. The first line of the matrix will be a vector $\vec{p_{1,r_1}} = [r_1, 0, 0, \cdots, 0]$ i.e only the first element is not zero and equal to r_1. We populate L_1 with hash values $H_{ros}(\vec{p_{1,r_1}})/r_1$. This way $r_1 \cdot H_{ros}(\vec{p_{1,r_1}})/r_1 = H_{ros}(\vec{p_{1,r_1}})$.

We then execute Wagner's algorithm which which will give use elements $c_1 \in L_1, c_2 \in L_2, \cdots, c_\ell \in L_\ell$ such that $c_1 + c_2 + \cdots + c_\ell = H_{ros}(\vec{p}_{\ell+1})$.

These elements will be of the form $c_1 = H_{ros}(\vec{p_{1,r_1}})/r_1$, $c_2 = H_{ros}(\vec{p_{2,r_2}})/r_2$, $\cdots, c_\ell = H_{ros}(\vec{p_{\ell,r_\ell}})/r_\ell$. We have now solved an ROS instance :

$$\begin{pmatrix} H_{ros}(\vec{p}_{1,r_1}) \\ H_{ros}(\vec{p}_{2,r_2}) \\ \vdots \\ H_{ros}(\vec{p}_{\ell,r_\ell}) \\ H_{ros}(\vec{p}_{\ell+1}) \end{pmatrix}_{(\ell+1)\times 1} \stackrel{?}{=} \begin{pmatrix} \vec{p}_1 = [r_1 & 0 & \cdots & 0] \\ \vec{p}_2 = [0 & r_2 & \cdots & 0] \\ \vdots & & & \\ \vec{p}_\ell = [0 & 0 & \cdots & r_\ell] \\ \vec{p}_{\ell+1} = [1 & 1 & \cdots & 1] \end{pmatrix}_{(\ell+1)\times \ell} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{pmatrix}_{\ell\times 1}$$

This is a well-known attack, however without serious threat against blind Schnorr or threshold Schnorr signatures due to the fact that it is of exponential cost. The threat of using Wagner's algorithm as a means of solving ROS instances is not really realistic.

5.3 Polynomial algorithm for the ROS problem

Although attacking ROS with Wagner's algorithm does not present a realistic threat due to its exponential complexity, the attack discussed in this section and published in [15] completely broke schemes who based their security on the hardness of the ROS problem. This attack did not only break schemes, but also raised questions on the validity of security proofs of schemes like [16] which can easily be broken by this algorithm but did not base their security on the ROS problem. We discuss further about those issues in Section 5.4. In

this section we present the polynomial algorithm to break the ROS problem. The intuition from where this algorithms stems from is the fact that the Wagner attack fixes the final vector of the matrix as $p_{\ell+1} = [1, 1, \cdots, 1]$ *; this is quite limiting. The authors of [15] wanted to allow the last line of the matrix to have different coefficients. This, as we will demonstrate in Section 5.5.3, allows for a very automated way to solve the ROS problem.*

We introduce the following notation: a polynomial $\mathbf{p} = p_0 + p_1 \cdot x_1 + \cdots + p_\ell \cdot x_\ell$ *has the following vector of coefficients :* $\vec{\mathbf{p}} = [p_1, \cdots, p_\ell]$*. The steps an adversary has to follow in order to produce an ROS solution are:*

1. *The adversary defines polynomials:* $\mathbf{p}_i^0 = x_i$*,* $\mathbf{p}_i^1 = 2 \cdot x_i$ *for* $i \in [\ell]$
2. *and* $c_i^b = 2^{-b} \cdot H_{ros}(\vec{\mathbf{p}}_i^b)$
3. *With overwhelming probability it will hold that* $c_i^0 \neq c_i^1$ *for all* $i \in [\ell]$*. Therefore, the adversary can define the following degree-1 polynomials:*

$$\mathbf{f}_i(x_i) = \frac{x_i - c_i^0}{c_i^1 - c_i^0}$$

   These polynomials have the following property: $\mathbf{f}_i(c_i^b) = b$*.*

4. *Define* $\mathbf{p}_{\ell+1} = \sum_{i=1}^{\ell} 2^{i-1} \cdot \mathbf{f}_i$ *which is also of degree-1 and can thus be written as*

$$\mathbf{p}_{\ell+1} = p_{\ell+1,0} + p_{\ell+1,1} \cdot x_1 + p_{\ell+1,2} \cdot x_2 + \cdots + p_{\ell+1,\ell} \cdot x_\ell.$$

5. *Define* $y = H_{ros}(\vec{\mathbf{p}}_{\ell+1}) + p_{\ell+1,0} = H_{ros}([p_{\ell+1,1}, p_{\ell+1,2}, \cdots, p_{\ell+1,\ell}]) + p_{\ell+1,0}$ *and find the binary representation of* $y = \sum_{i=1}^{\ell} 2^{i-1} \cdot b_i$*.*
6. *The adversary outputs the solution*

$$\begin{pmatrix} \vec{\mathbf{p}}_1^{b_1} \\ \vec{\mathbf{p}}_2^{b_2} \\ \vdots \\ \vec{\mathbf{p}}_\ell^{b_\ell} \\ \vec{\mathbf{p}}_{\ell+1} \end{pmatrix} \quad \text{and} \quad \vec{c} = [c_1^{b_1}, c_2^{b_2}, \cdots, c_\ell^{b_\ell}].$$

For $i \in [\ell]$ *it holds that* $\langle \vec{\mathbf{p}_i^{\mathbf{b_i}}}, \boldsymbol{c} \rangle = 2^{b_i} \cdot c_i^{b_i} = H_{ros}(\vec{\mathbf{p}_i^{\mathbf{b_i}}})$ *and* $\langle \vec{\mathbf{p}_{\ell+1}}, \vec{c} \rangle = \mathbf{p}_{\ell+1}(\vec{c}) - p_{\ell+1} = \sum_{i=1}^{\ell} 2^{i-1} \cdot \mathbf{f}_i(c_i^{b_i}) - p_{\ell+1,0} = \sum_{i=1}^{\ell} 2^{i-1} \cdot b_i - p_{\ell+1,0} = H_{ros}(\vec{\mathbf{p}_{\ell+1}})$

5.4 The InsecureMusig scheme

One of the multi-signature schemes affected by this algorithm was the scheme presented in [16]. This was considered a state-of-the-art multi-signature schemes as it only consisted of 2 rounds which was the lowest amount of signing rounds until then. Apart from that, it was suggested that this signing scheme be added to Bitcoin in order to improve performance

and user privacy. Thankfully, this did not happen ; had it happened, it could have been a massive security flaw that could have caused blockchain theft. The scheme was presented to be provably secure in the plain public-key model ; meaning that signers are only required to have a public key, but do not have to prove knowledge of the private key corresponding to their public key before engaging in the protocol. The reason why this was allowed is because the group's public key is calculated via a non-trivial key aggegation algorithm which does not allow for rogue-key attacks.

5.4.1 Rogue-key attacks

Rogue-key attacks [47] are a form of attack where an adversary in able to influence the outcome of a group key generation algorithm to a value he wants. Let's assume that the aggregated public key of a discrete logarithm multi-signature scheme is calculated as $pk = \prod_{i\in[n]} pk_i$ *where* pk_i *is the public key of each member involved in the protocol. An adversary can easily influence the final group public key to result in a value X for which he knows the discrete logarithm x by just waiting for the first* $n-1$ *members to reveal their keys* $p_1, p_2, \cdots, p_{n-1}$ *and then reveal his own key as* $pk_n = X \cdot \prod_{i\in[n-1]} pk_i$*. By doing that, it is evident that the resulting group public key is* $pk = X$ *for which the corresponding private key (the discrete logarithm) is known to the adversary. In such case, the adversary could produce signatures on behalf of the group for any message by following the single-party signing algorithm (Schnorr in this case).*

5.4.2 InsecureMusig key aggregation against rogue-key attacks

The authors of the scheme suggested a different way to aggregate each parties keys in order to avoid such attacks. The new key aggregation algorithm works as follows: assume that each member has a public, private key pair $(pk_i, sk_i) = (X_i = g^{x_i}, x_i)$ *and they reveal their public key to the rest of the group. After everyone has revealed their public keys* $pk_i = X_i$*, the key pairs will change so that the final key pair of each individual is tied to the other participants' keys. The way this happens is that a binding factor* $a_i = H(L, X_i)$ *is calculated by each participant (where L is the list of all public keys before the transformation takes place i.e* $L = \{X_1 = g^{x_1}, X_2 = g^{x_2}, \cdots, X_n = g^{x_n}\}$*. After that, the final keys which will be used by each member are* $(pk_i, sk_i) = (X_i' = g^{a_i \cdot x_i}, a_i \cdot x_i)$*. The final public key of the group is* $pk = \prod_{i\in[n]} X_i' = \prod_{i\in[n]} X_i^{a_i}$*.*

The reason this key aggregation algorithm works at avoiding rogue-key attacks is because the final public key of the group key is tied to the binding factors which are themselves tied to the original public keys revealed by the members. It is impossible to calculate the final public key of a member before all the rest have revealed their initial public keys, since in that case the binding factors cannot be calculated.

5.4.3 Algorithms of the InsecureMusig scheme

We remind that a multi-signature scheme M *consists of algorithms* $(Setup, KeyGen, (Sign_1, Sign_2, \cdots, Sign_r), Combine, Verify)$ *(since InsecureMusig is a two-round protocol we*

have $r = 2$). We add another algorithm KeyAgg for clarity but in reality we can consider that key aggregation happens during the two-step KeyGen algorithm (first step is initial key generation and second step is adding the binding factors to each key pair and calculating the group's public key). We describe the algorithms:

- *Setup. We work in a group $\mathbb{G}$ of order p and g is a generator of the group. The scheme will also need two hash functions $H_0, H_1 : \{0,1\}^* \to \mathbb{Z}_p$.*
- *KeyGen. Each signer generates a random private key $x \xleftarrow{\$} \mathbb{Z}_p$ and computes the corresponding public key $X = g^x$.*
- *KeyAgg. After all initial public keys have been revealed and added to the list $L = \{X_1, \cdots, X_n\}$, the actual key pair of each participant P_i are calculated as $(pk_i, sk_i) = (X_i^{a_i}, x_i \cdot a_i)$ where $a_i = H_0(L, X_i)$ and the aggregated public key is $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$.*
- *Sign$_1$: We assume the signers wish to sign an (agreed-on) message m. Each signer P_i generates a random $r_i \xleftarrow{\$} \mathbb{Z}_p$, computes $R_i = g^{r_i}$ and sends R_i to all other cosigners.*
- *Sign$_2$: After receiving all $\{R_i\}_{i\in[n]}$ from all other cosigners, participant P_i computes the group commitment $R = \prod_{i\in[n]} R_i$ the challenge of the Schnorr signature $c = H_1(\tilde{X}, R, m)$ and his partial signature share $z_i = r_i + c \cdot a_i \cdot x_i$ which he sends out to all other signers (or the central coordinator node).*
- *Combine. After all partial signatures $\{z_i\}$ have been sent out the final group signature on m is produced as $\sigma = (R, \sum_{i\in[n]} z_i)$.*
- *Verify. Any public validator can verify the signature under the public key $\tilde{X}$ by using the Schnorr single party verification algorithm on $\sigma = (R, z = \sum_{i\in[n]} z_i)$.*

We provide a quick correctness proof for this scheme before discussing some important information.

Theorem 5.1. *Correctness of InsecureMusig The scheme M = InsecureMusig follows the definition of correctness from Section 4.3.1.*

Proof. The final group response is $z = \sum_{i\in[n]} z_i = \sum_{i\in[n]} r_i + c \cdot a_i \cdot x_i$ and we also have $R = \prod_{i\in[n]} R_i$ and $c = H_1(\tilde{X}, R, m)$.
It holds that: $g^z = g^{\sum_{i\in[n]} r_i + ca_ix_i} = g^{\sum_{i\in[n]} r_i} \cdot g^{\sum_{i\in[n]} caixi} = (\prod_{i\in[n]} g^{r_i}) \cdot (\prod_{i\in[n]} X_i^{c\cdot a_i}) = R \cdot \tilde{X}^c$ □

As discussed in earlier sections, the final group signature looks exactly like a single-party Schnorr signature and can be verified under a common, aggregated group key. Another key thing to note is that each partial signature $\sigma_i = (R_i, z_i)$ can be verified by any member that has access to R_i and z_i. This partial signature is also exactly identical to a Schnorr signature that can be verified under the participants public key $X_i^{a_i}$. This is really important in detecting adversarial members who might try to stop the group signature from verifying.

Setup(1^κ)

$(G, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

$H_0, H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p$

$par \leftarrow ((\mathbb{G}, p, g), H_0, H_1)$

return par

KeyGen(par)

for j = 1 **to** n:

$x_j \leftarrow \mathbb{Z}_p, \quad X_j \leftarrow g^{x_j}$

$pk_j \leftarrow X_j$

$sk_j \leftarrow x_j$

return $(\{pk\}_{j\in[n]}, \{sk\}_{j\in[n]})$

KeyAgg$(\{X_j\}_{j\in SS})$

for i = 1 **to** n

$a_i \leftarrow H_0((X_1, \ldots, X_n), X_i)$

$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i^{a_i}$

return $\tilde{X}$

Sign$_1(SS, m, i, \{pm_{0,j}\}_{j\in SS}, sk_i, st_i)$

$r_i \leftarrow \mathbb{Z}_p$

$R_i \leftarrow g^{r_i}$

$pm_{1,i} \leftarrow R_i$

$st_i \leftarrow r_i$

return $pm_{1,i}, st_i$

Sign$_2(SS, m, i, \{pm_{1,j}\}_{j\in SS}, sk_i, st_i)$,

parse $r_i, \leftarrow st_i, x_i \leftarrow sk_i$

parse $\{R_j\}_{j\in SS} \leftarrow \{pm_{1,j}\}_{j\in SS}$

$a_i \leftarrow H_0((X_1, \cdots, X_n), X_i)$

$R \leftarrow \prod_{j=1}^{n} R_j$

$c \leftarrow H_1(R, \tilde{X}, m)$

$z_i \leftarrow r_i + c \cdot a_i \cdot x_i$

$pm_{2,i} \leftarrow z_i$

return $pm_{2,i}, st_i$

Combine$(\{pm_{k,j}\}_{k\in[2],j\in SS})$

parse $\{R_j\}_{j\in SS} \leftarrow \{pm_{1,j}\}_{j\in SS}$

parse $\{z_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

$R \leftarrow \prod_{j\in SS} R_j$

$c \leftarrow H_1(R, \tilde{X}, m)$

for j = 1 **to** n

$a_j \leftarrow H_0((X_1, \ldots, X_n), X_j)$

if $g^{z_j} \neq R_j \cdot X_j^{a_j \cdot c}$

return $\perp$

$z \leftarrow \sum_{i=1}^{n} z_i$

$R \leftarrow \prod_{i=1}^{n} R_i$

$\sigma \leftarrow (R, z)$

return σ

Verify(σ, Y, m)

parse $\sigma \leftarrow (R, z)$

$c \leftarrow H_1(R, Y, m)$

return 1 **if** $g^z = R \cdot Y^c$, **else** 0

Figure 5.2: *The InsecureMusig scheme*

5.5 Breaking InsecureMusig

As we mentioned earlier, this scheme was considered state-of-the-art as it only required two rounds to output a signature. However, its success was short-lived as two separate practical attacks can break the scheme. This was extremely surprising, as the InsecureMusig scheme had a security proof which relied on the OMDL assumption discussed in Section 3.2.2. In fact, one of the main selling points of this scheme is that it was considered provably secure and that is why it was been considered as a viable candidate to be added to blockchains. Clearly, the proof was not correct. We discuss shortly about the issues of the proof in section Section 5.5.4 and we refer the reader to [17] for a much more detailed discussion.

5.5.1 Drijvers attack

The first attack on the InsecureMusig scheme appeared in [17] and used Wagner's Generalized Birthday attack in order to create a one-more forgery against the scheme when an adversary was able to open multiple signing sessions with an honest signing party. The first thing to notice about the InsecureMusig scheme is that it went to great lengths to avoid rogue-key attacks but did not bother protecting against rogue-commitment attacks i.e the adversary can fully control the commitment R of the group signature. The way he can do this is by waiting for all honest signers to reveal their $\{R_i\}_{i\in Hon}$ during the first signing round and then reveal his commitment as $T \cdot \prod_{i\in Hon} R_i^{-1}$; it is obvious that the resulting group commitment is $R = T$. For simplicity we will assume that we only have 2 signers, Bob (an honest party) and the adversary. At the end of the key generation phase, Bob has keys (pk_B, sk_B) which we can assume to be of the form $(g^{x\cdot a}, x \cdot a)$ were a is the binding factor (the attack is exactly the same if we are to assume that the keys are (g^x, x), the difference is only notational).

The adversary will open $\ell-1$ parallel signing sessions with Bob for messages $m_1, \cdots, m_{\ell-1}$ which Bob agrees on signing. For each signing session, Bob will send his commitment nonce $\{R_i\}_{i\in[\ell-1]}$. Before sending out his commitment nonces for each session, the adversary will perform an instance of the k-sum (with $k = \ell$) algorithm described in Section 5.2 to find specifically calculated group nonces that will allow him to mount the attack.

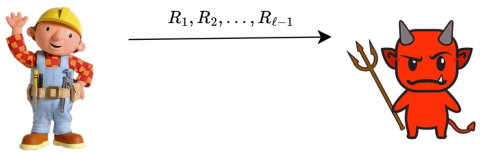

Figure 5.3: *Parallel sessions*

We now explain how the adversary populates his ℓ lists: Lists $L_1, \ldots, L_{\ell-1}$ will be populated with elements of the form $H_0(g^r, m_k, \tilde{X})$ for many r values (m_k is the message signed on session k and $\tilde{X}$ is the public key of the group). The elements of each list depict challenge values of the session for the resulting group signature because the adversary will use $R_k^{-1} \cdot g^{r_k}$ as his commitment nonce for session k. List L_ℓ will be populated with values of the form $-H_0(\prod_{k=1}^{\ell-1} R_k, m^*, \tilde{X})$ for many m^* values.

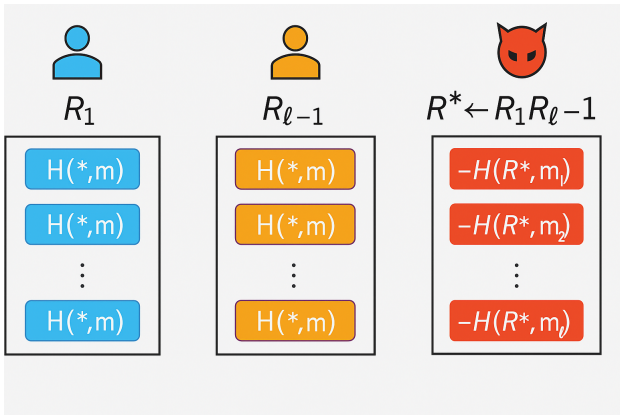

Figure 5.4: *List population for the Drijvers attack*

Then the adversary runs Wagner's algorithm on theses ℓ lists. The algorithm will find values $c_1 \in L_1, \ldots, c_{\ell-1} \in L_{\ell-1}, c_\ell \in L_\ell$ such that $c_1 + \cdots + c_{\ell-1} + c_\ell = 0 \iff H(g^{r_1}, \tilde{X}, m_1) + \ldots H(g^{r_{\ell-1}}, \tilde{X}, m_{\ell-1}) = H(\prod_{k=0}^{l-1} R_k, \tilde{X}, m^*)$. The adversary sends his commitment nonces $R_1^{-1} \cdot g^{r_1}, R_2^{-1} \cdot g^{r_2}, \cdots, R_{\ell-1}^{-1} \cdot g^{r_{\ell-1}}$ back to Bob. Each one of those nonces will lead to the target challenge value c_k for session k. Bob will reply with a valid partial signature share z_k for session k which will verify under his public key pk_B. Thus, it will hold that $g^{z_k} = R_k \cdot pk_B{}^{c_k}, k \in [\ell - 1]$.

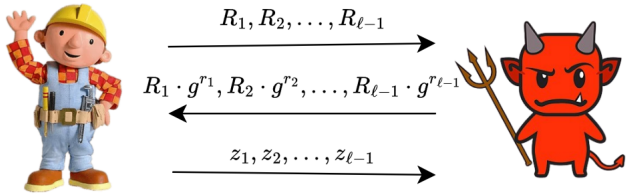

Figure 5.5: *Parallel sessions response*

The adversary then defines:

- $R^* = \prod_{k=1}^{\ell-1} R_k$
- $c^* = -c_\ell = H(\prod_{k=1}^{\ell-1} R_k, m^*, pk) = H(R^*, m^*, pk)$

- $z^* = \sum_{k=1}^{\ell-1} z_k$

It holds that:
$g^{z^*} = g^{\sum_{k=1}^{\ell-1} z_k} = \prod_{k=1}^{\ell-1} g^{z_k} = \prod_{k=1}^{\ell-1} R_k \cdot pk_B{}^{c_k} = (\prod_{k=1}^{\ell-1} R_k) \cdot (\prod_{k=1}^{\ell-1} pk_B{}^{c_k}) = R^* \cdot pk_B^{\sum_{k=1}^{\ell-1} c_k} = R^* \cdot pk_B{}^{c^*}$
And therefore $\sigma^* = (R^*, z^*)$ is a valid signature share of Bob's on m^*.
The adversary has therefore succeeded in forging a signature under Bob's key ; he can just add his own signature share on Bob's partial signature and create a forgery that verifies under the aggregated public key of the group.
Notice that this attack can easily be extended when the honest participants are more than one ; the adversary just waits for all of them to send their commitment nonces for each session and the attack is exactly the same if we replace Bob's commitment share with $R_k = \prod_{i \in Hon} R_{k,i}$ where $R_{k,i}$ is the commitment nonce of participant i for session k.
The Drijvers attack, although exponential, is practical: with 127 parallel sessions we can get a forgery in 2^{45} steps. However, the polynomial ROS attack that we presented can break schemes even faster with less computation.

5.5.2 Polynomial ROS attack

In order to use the polynomial ROS attack against InsecureMusig, the adversary will once again need to open multiple parallel signing sessions with the honest participants (again we limit the honest participants to one, Bob, but the attack can easily be generalized. This step is the same as in Figure 5.3. For clarity we will symbolize Bob's commitment shares with D_i instead of R_i. Before sending out his commitment nonces, the adversary uses the algorithms steps as follows:

1. *The adversary samples $r_{i,0}$ and $r_{i,1}$ and computes $R_i^0 = g^{r_{i,0}} \cdot D_i$ and $R_i^0 = g^{r_{i,0}} \cdot D_i$ and the challenges $c_i^0 = H_0(R_i^0, m_i, \tilde{X})$ and $c_i^1 = H_0(R_i^1, m_i, \tilde{X})$.*

2. *With overwhelming probability it holds that $c_i^0 \neq c_i^1$ so the polynomial $\mathbf{p} = \sum_{i=1}^{\ell-1} 2^{i-1} \cdot \frac{x_i - c_i^0}{c_i^1 - c_i^0}$ can be defined.*

3. *The adversary calculates $R_\ell = \prod_{i=1}^{\ell-1} D_i^{p_i}$ and $c_\ell = H_0(R_\ell, m_\ell, \tilde{X})$ and finds the binary representation of $c_\ell + p_0 = \sum_{i=1}^{\ell-1} 2^{i-1} \cdot b_i$.*

4. *For each session k, the adversary proceeds by sending his commitment nonce $D_k^{-1} \cdot R_k^{b_k}$ to Bob.*

5. *After Bob replies with his partial signature shares z_k, the adversary outputs $(R_\ell = \prod_{i=1}^{\ell-1} D_i^{p_i}, z_\ell = \sum_{i=1}^{\ell-1} z_i \cdot p_i)$.*

It holds that $g^{z_\ell} = g^{\sum_{i=1}^{\ell-1} z_i \cdot p_i} = g^{\sum_{i=1}^{\ell-1} p_i \cdot (d_i + c_i^{b_i} \cdot sk_B)} = (g^{\sum_{i=1}^{\ell-1} p_i \cdot d_i}) \cdot (g^{sk_B \cdot \sum_{i=1}^{\ell-1} p_i \cdot c_i^{b_i}}) = R_\ell \cdot g^{sk_B \cdot [\mathbf{p}(\vec{\mathbf{c}}) - p_0]} = R_\ell \cdot pk_B^{c_\ell}$

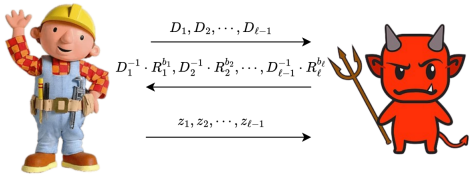

Figure 5.6: *Parallel sessions response in the ROS algorithm*

5.5.3 Implementation of the ROS attack against InsecureMusig

As part of this thesis, we implement the polynomial ROS attack in SageMath using the elliptic curve parameters of Bitcoin [48]. Our code can forge a signature in about 10 *seconds showcasing the practicality of the attack and the serious threat that it poses. We use SHA256 [31] as* H_0*. Some key parts of the code include:*

```
Zq = GF(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F)
E = EllipticCurve(Zq, [0, 7])
G = E.lift_x(0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798)
p = G.order()
Zp = GF(p)

# Bob's keys
x = Zp.random_element()
X = G * int(x)
```

Figure 5.7: *Parameter generation for the attack including the elliptic curve parameters and the key pair of the honest signer.*

```
# Hash function
def hash_to_Zp(data):
    h = hashlib.sha256(str(data).encode()).hexdigest()
    return Zp(int(h, 16))

# Function to verify Bob's partial signatures
def verifyBob(message, R_bob, R_group, z):
    chall = hash_to_Zp((R_group, message))
    return (G * int(z) == (R_bob + X * int(chall)))

# Function to verify a normal Schnorr signature
def verify(message, R , z):
    chall = hash_to_Zp((R, message))
    return (G * int(z) == (R + X * int(chall)))

def inner_product(coefficients, values):
    return sum(y*int(x) for x, y in zip(coefficients, values))
```

Figure 5.8: *Helper functions including the hash function* H_0*, a function to verify partial signature shares of the honest signer, a function verify Schnorr signatures and a a function to compute inner products of vectors.*

```python
# Hash function
def hash_to_Zp(data):
    h = hashlib.sha256(str(data).encode()).hexdigest()
    return Zp(int(h, 16))

# Function to verify Bob's partial signatures
def verifyBob(message, R_bob, R_group, z):
    chall = hash_to_Zp((R_group, message))
    return (G * int(z) == (R_bob + X * int(chall)))

# Function to verify a normal Schnorr signature
def verify(message, R , z):
    chall = hash_to_Zp((R, message))
    return (G * int(z) == (R + X * int(chall)))

def inner_product(coefficients, values):
    return sum(y*int(x) for x, y in zip(coefficients, values))
```

Figure 5.9: *Interaction between the adversary and the honest signer and forgery creation.*

In our code we use 256 *parallel session between Bob and the adversary. The entire code can be found at The full source code is available at our GitHub repository.*

5.5.4 Insecure shortcuts in InsecureMusig

It is evident from the previous attacks that the InsecureMusig cannot provide concurrent security (that is, security when parallel sessions are allowed). However, as we mentioned earlier, the InsecureMusig scheme came with a proof that claimed that it was unforgeable under the OMDL assumption. In [17] it was shown that this proof was flawed. The issue was in the amount of DL oracle accesses that the reduction had. The point is that the reduction must have t *accesses to a DL oracle and provide* $t + 1$ *discrete logarithms of* $t + 1$ *challenges. In the original proof of InsecureMusig the reduction was able to make more than* t *queries to the oracle thus trivially solving the* $t + 1$ *discrete logarithms. Not only that, but Drijvers et al showed that the InsecureMusig scheme is impossible to be proven secure under the DL or OMDL assumptions with any currently known technique.*

Chapter 6

The MuSig1 and MuSig2 schemes and concurrently secure techniques

6.1 Concurrent security challenges

A very simple approach to add concurrent security to InsecureMusig is to completely avoid it by not allow parallel sessions. However, this would render the scheme impractical for many real world applications. Instead, understanding why the attacks of the previous chapter work is vital to the construction of secure multi- and threshold signing schemes. As we mentioned before, the adversary can have total control of the group's commitment with the way that InsecureMusig was constructed. This is the main issue behind the attacks and is a very important point to be fixed. Fixing it however, could be costly. Overall, throughout the literature we can find two ways to combat this.

1. ***Extra commitment round**: Add an extra commitment round on the commitment nonces. This way after the nonces have been revealed we can check whether they match the commitments of the previous round and disqualify signers for which it does not.*

2. ***Binding factor**: Add an extra factor to each commitment nonce so that each nonce depends on all other nonces. This way the final nonces cannot be computed until they have all been revealed. This is very similar to the key aggregation technique of InsecureMusig to avoid rogue-key attacks.*

In this chapter we describe two concurrently secure schemes that were created in order to add security to the InsecureMusig scheme. Those schemes are MuSig1 [18] and MuSig2 [19]. These schemes use the two techniques that we mentioned with MuSig1 using the extra commitment round whereas MuSig2 the binding factor technique. In Section 6.2 we describe the MuSig1 scheme, in Section 6.3 MuSig2 is presented. Finally, in Section 6.4 we compare the two schemes and their techniques.

6.2 The MuSig1 scheme

MuSig1 [18] is the second version of the InsecureMusig paper [16] and is thus quite similar. The only difference between the two schemes is that MuSig1 is a tree-round scheme

because it adds an extra commitment round at the start of every signing session.

6.2.1 Algorithms of the MuSig1 scheme

The algorithms are very similar to the ones in Section 5.4.3:

- *Setup. We work in a group $\mathbb{G}$ of order p and g is a generator of the group. The scheme will also need three hash functions $H_{agg}, H_{com}, H_{sig} : \{0,1\}^* \to \mathbb{Z}_p$.*
- *KeyGen. Each signer generates a random private key $x \xleftarrow{\$} \mathbb{Z}_p$ and computes the corresponding public key $X = g^x$.*
- *KeyAgg. After all initial public keys have been revealed and added to the list $L = \{X_1, \cdots, X_n\}$, the actual key pair of each participant P_i are calculated as $(pk_i, sk_i) = (X_i^{a_i}, x_i \cdot a_i)$ where $a_i = H_{agg}(L, X_i)$ and the aggregated public key is $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$.*
- *$Sign_1$: We assume the signers wish to sign an (agreed-on) message m. Each signer P_i generates a random $r_i \xleftarrow{\$} \mathbb{Z}_p$, computes $R_i = g^{r_i}$, $t_i = H_{com}(R_i)$ and sends out t_i to all other signers.*
- *$Sign_2$: After receiving all $\{t_i\}_{i\in[n]}$ from all other cosigners, participant P_i reveals his commitment nonce R_i by sending it out to all signers.*
- *$Sign_3$: After receiving all $\{R_i\}_{i\in[n]}$ from all other cosigners, participant P_i checks that $t_i = H_{com}(R_i)$ and aborts the protocol if this is not the case. Otherwise, he computes the group commitment $R = \prod_{i\in[n]} R_i$, the challenge of the Schnorr signature $c = H_{sig}(\tilde{X}, R, m)$ and his partial signature share $z_i = r_i + c \cdot a_i \cdot x_i$ which he sends out to all other signers (or the central coordinator role).*
- *Combine. After all partial signatures $\{z_i\}$ have been sent out the final group signature on m is produced as $\sigma = (R, \sum_{i\in[n]} z_i)$.*
- *Verify. Any public validator can verify the signature under the public key $\tilde{X}$ by using the Schnorr single party verification algorithm on $\sigma = (R, z = \sum_{i\in[n]} z_i)$.*

The correctness is identical to the one presented in Section 5.4.3.

It is clear that the attacks of the previous chapter cannot be used here as an adversary cannot wait for the honest signers to reveal their commitment nonces before revealing his own as if his commitment hash from the first round does not check out with the commitment nonce he reveals during the second round, the protocol aborts. Therefore, the adversary is unable to adaptively choose his commitment nonce.

6.2.2 Complexity analysis of MuSig1

We use the metrics described in Section 4.6 to measure the complexity of the MuSig1 scheme. We do not count the cost of key generation KeyGen and key aggregation KeyAgg as these do not occur for every session.

First of all, each session consists of 3 rounds.

For each session, each signer has send out elements of size:

Setup(1^κ)

$(G, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

$H_{agg}, H_{com}, H_{sig} : \{0,1\}^* \rightarrow \mathbb{Z}_p$

$par \leftarrow ((\mathbb{G}, p, g), H_{agg}, H_{com}, H_{sig})$

return par

KeyGen(par)

for j = 1 **to** n:

$x_j \leftarrow \mathbb{Z}_p, \quad X_j \leftarrow g^{x_j}$

$pk_j \leftarrow X_j$

$sk_j \leftarrow x_j$

return $(\{pk\}_{j\in[n]}, \{sk\}_{j\in[n]})$

KeyAgg$(\{X_j\}_{j\in SS})$

for i = 1 **to** n

$a_i \leftarrow H_{agg}((X_1, \ldots, X_n), X_i)$

$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i^{a_i}$

return $\tilde{X}$

Sign$_1(SS, m, i, \{pm_{0,j}\}_{j\in SS}, sk_i, st_i)$

$r_i \leftarrow \mathbb{Z}_p$

$R_i \leftarrow g^{r_i}$

$t_i \leftarrow H_{com}(R_i)$

$pm_{1,i} \leftarrow t_i$

$st_i \leftarrow (r_i, R_i)$

return $pm_{1,i}, st_i$

Sign$_2(SS, m, i, \{pm_{1,j}\}_{j\in SS}, sk_i, st_i)$,

parse $(R_i, r_i,) \leftarrow st_i$

parse $\{t_j\}_{j\in SS} \leftarrow \{pm_{1,j}\}_{j\in SS}$

$pm_{2,i} \leftarrow R_i$

$st_i \leftarrow (R_i, r_i, \{t_j\}_{j\in SS})$

return $pm_{2,i}, st_i$

Sign$_3(SS, m, i, \{pm_{2,j}\}_{j\in SS}, sk_i, st_i)$,

parse $(R_i, r_i, \{t_j\}_{j\in SS}) \leftarrow st_i, x_i \leftarrow sk_i$

parse $\{R_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

for j = 1 **to** n

if $H_{com}(R_j) \neq t_j$

return $\perp$

$a_i \leftarrow H_{agg}((X_1, \cdots, X_n), X_i)$

$R \leftarrow \prod_{j=1}^{n} R_j$

$c \leftarrow H_1(R, \tilde{X}, m)$

$z_i \leftarrow r_i + c \cdot a_i \cdot x_i$

$pm_{3,i} \leftarrow z_i$

return $pm_{3,i}, st_i$

Combine$(\{pm_{k,j}\}_{k\in[3],j\in SS})$

parse $\{R_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

parse $\{z_j\}_{j\in SS} \leftarrow \{pm_{3,j}\}_{j\in SS}$

$R \leftarrow \prod_{j\in SS} R_j$

$c \leftarrow H_{sig}(R, \tilde{X}, m)$

for j = 1 **to** n

$a_j \leftarrow H_{agg}((X_1, \ldots, X_n), X_j)$

if $g^{z_j} \neq R_j \cdot X_j^{a_j \cdot c}$

return $\perp$

$z \leftarrow \sum_{i=1}^{n} z_i$

$R \leftarrow \prod_{i=1}^{n} R_i$

$\sigma \leftarrow (R, z)$

return σ

Verify(σ, Y, m)

parse $\sigma \leftarrow (R, z)$

$c \leftarrow H_1(R, Y, m)$

return 1 **if** $g^z = R \cdot Y^c$, **else** 0

Figure 6.1: *The MuSig1 scheme*

- ***Sign***$_1$: $n \cdot \mathbb{Z}_p$ *(broadcast)*
- ***Sign***$_2$: $n \cdot \mathbb{G}$ *(broadcast)*
- ***Sign***$_3$: $\mathbb{Z}_p$ *(sent to coordinator)*

Thus, the total amount of communication for a MuSig1 session is: $n \cdot ((n+1) \cdot \mathbb{Z}_p + n \cdot \mathbb{G})$. *For each session, each signer has to perform computations of cost:*

- ***Sign***$_1$: $GExp + Hh$
- ***Sign***$_2$: -
- ***Sign***$_3$: $(n+2) \cdot Hh + (n-1) \cdot GMul + 2 \cdot SMul$

Therefore, the total amount of computation per signer is: $GExp + (n+3) \cdot Hh + (n-1) \cdot GMul + 2 \cdot SMul$.

6.3 The MuSig2 scheme

The only difference between MuSig1 [18] and InsecureMusig [16] is that MuSig1 had to add an extra commitment round. This adds minimal computation cost, however the rounds of the protocol get increased from 2 to 3. The creators of MuSig2 [19] wanted to create a 2 round concurrently secure multi-signature scheme. To do so, they used the second technique described in Section 6.1 and two nonces per signer instead of just one.

6.3.1 Algorithms of the MuSig2 scheme

- *Setup. We work in a group* $\mathbb{G}$ *of order* p *and* g *is a generator of the group. The scheme will also need three hash functions* $H_{agg}, H_{non}, H_{sig} : \{0,1\}^* \to \mathbb{Z}_p$.
- *KeyGen. Each signer generates a random private key* $x \xleftarrow{\$} \mathbb{Z}_p$ *and computes the corresponding public key* $X = g^x$.
- *KeyAgg. After all initial public keys have been revealed and added to the list* $L = \{X_1, \cdots, X_n\}$, *the actual key pair of each participant* P_i *are calculated as* $(pk_i, sk_i) = (X_i^{a_i}, x_i \cdot a_i)$ *where* $a_i = H_{agg}(L, X_i)$ *and the aggregated public key is* $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$.
- *Sign*$_1$: *We assume the signers wish to sign an (agreed-on) message* m. *Each signer* P_i *generates two random values* $r_{i,1}, r_{i,2} \xleftarrow{\$} \mathbb{Z}_p$, *computes* $R_{i,1} = g^{r_{i,1}}$, $R_{i,2} = g^{r_{i,2}}$ *and sends out* $R_{i,1}, R_{i,2}$ *to all other signers.*
- *Sign*$_2$: *After receiving all* $\{R_{i,j}\}_{i\in[n],j\in[2]}$ *from all other cosigners, participant* P_i *computes a binding factor* $b = H_{non}(\tilde{X}, (R_{j,1})_{j\in[n]}, (R_{j,2})_{j\in[n]}, m)$, *the group commitment* $R = \prod_{j\in[n]} R_{j,1} \cdot R_{j,2}^b$, *the challenge of the Schnorr signature* $c = H_{sig}(\tilde{X}, R, m)$ *and his partial signature share* $z_i = r_{i,1} + r_{i,2} \cdot b + c \cdot a_i \cdot x_i$ *which he sends to the coordinator.*
- *Combine. After all partial signatures* $\{z_i\}$ *have been sent out the final group signature on* m *is produced as* $\sigma = (R, \sum_{i\in[n]} z_i)$.

- *Verify. Any public validator can verify the signature under the public key $\tilde{X}$ by using the Schnorr single party verification algorithm on $\sigma = (R, z = \sum_{i\in[n]} z_i)$.*

Once again, each partial signature share is a Schnorr signature and its validity can be checked under the public key of each participant.

6.3.2 Escaping ROS attacks with two nonces

The binding factor solution makes it so that each participant's commitment nonce is dependent on the nonces of all other participants, since the binding factor involves the nonces of the entire signing set. The important thing is that we have to use two nonces per signer instead of just one. Recall that in the original InsecureMusig paper, the adversary could perform his attack by calculating R_i such that:

$$\sum_{k=1}^{\ell-1} \mathrm{H}_{sig}(\widetilde{X}, R_k, m_k) = \mathrm{H}_{sig}(\widetilde{X}, R^*, m^*) \tag{6.1}$$

With the usage of the binding factor, it is tempting to fall back to only a single nonce and rely on each signer having a commitment share $\hat{R}_i = R_i^b$. This would however fail as the adversary can effectively eliminate b by redefining $R^* = \prod_{k=1}^{\ell-1} R_k$ and considering the equation:

$$\sum_{k=1}^{\ell-1} \frac{\mathrm{H}_{sig}(\widetilde{X}, R_k^{b_k}, m_k)}{b_k} = \mathrm{H}_{sig}(\widetilde{X}, R^*, m^*) \tag{6.2}$$

We recommend reading [19] for a more detailed description of the one-nonce problem.

6.3.3 Complexity analysis of MuSig2

Each session consists of 2 rounds. For each session, each signer has to send out elements of size:

- ***Sign***$_1$*:* $n \cdot 2 \cdot \mathbb{G}$ *(broadcast)*
- ***Sign***$_2$*:* $\mathbb{Z}_p$ *(sent to coordinator)*

Thus, the total amount of communication for a MuSig2 sessions is : $n \cdot (n \cdot 2 \cdot \mathbb{G} + \mathbb{Z}_p)$. *For each session, each signer has to perform computations of cost:*

- ***Sign***$_1$*:* $2 \cdot GExp$
- ***Sign***$_2$*:* $3 \cdot Hh + 2 \cdot n \cdot GMul + GExp + 3 \cdot SMul$

Therefore, the total amount of computation for each signer is $3 \cdot Hh + 2 \cdot n \cdot GMul + 3 \cdot GExp + 3 \cdot SMul$.

Setup(1^κ)

$(G, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

$H_{agg}, H_{non}, H_{sig} : \{0,1\}^* \rightarrow \mathbb{Z}_p$

$par \leftarrow ((\mathbb{G}, p, g), H_{agg}, H_{non}, H_{sig})$

return par

KeyGen(par)

for j = 1 **to** n:

$x_j \leftarrow \mathbb{Z}_p, \quad X_j \leftarrow g^{x_j}$

$pk_j \leftarrow X_j$

$sk_j \leftarrow x_j$

return $(\{pk\}_{j\in[n]}, \{sk\}_{j\in[n]})$

KeyAgg$(\{X_j\}_{j\in SS})$

for i = 1 **to** n

$a_i \leftarrow H_{agg}((X_1, \ldots, X_n), X_i)$

$\tilde{X} \leftarrow \prod_{i=1}^n X_i^{a_i}$

return $\tilde{X}$

Sign$_1(SS, m, i, \{pm_{0,j}\}_{j\in SS}, sk_i, st_i)$

$r_{i,1}, r_{i,2} \leftarrow \mathbb{Z}_p$

$R_{i,1} \leftarrow g^{r_{i,1}}$

$R_{i,2} \leftarrow g^{r_{i,2}}$

$pm_{1,i} \leftarrow (R_{i,1}, R_{i,2})$

$st_i \leftarrow (r_{i,1}, r_{i,2})$

return $pm_{1,i}, st_i$

Sign$_2(SS, m, i, \{pm_{1,j}\}_{j\in SS}, sk_i, st_i)$,

parse $(r_{i,1}, r_{i,2}) \leftarrow st_i, x_i \leftarrow sk_i$

parse $\{R_{j,u}\}_{j\in SS, u\in[2]} \leftarrow \{pm_{2,j}\}_{j\in SS}$

$b \leftarrow H_{non}(\tilde{X}, (R_{j,1})_{j\in[n]}, (R_{j,2})_{j\in[n]}, m)$

$a_i \leftarrow H_{agg}((X_1, \cdots, X_n), X_i)$

$R \leftarrow \prod_{j=1}^n R_{j,1} \cdot R_{j,2}^b$

$c \leftarrow H_{sig}(R, \tilde{X}, m)$

$z_i \leftarrow r_{i,1} + r_{i,2} \cdot b + c \cdot a_i \cdot x_i$

$pm_{3,i} \leftarrow z_i$

return $pm_{2,i}, st_i$

Combine$(\{pm_{k,j}\}_{k\in[2], j\in SS})$

parse $\{R_{j,u}\}_{j\in SS, u\in[2]} \leftarrow \{pm_{1,j}\}_{j\in SS}$

parse $\{z_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

$b \leftarrow H_{non}(\tilde{X}, (R_{j,1})_{j\in[n]}, (R_{j,2})_{j\in[n]}, m)$

$R \leftarrow \prod_{j=1}^n R_{j,1} \cdot R_{j,2}^b$

$c \leftarrow H_{sig}(R, \tilde{X}, m)$

for j = 1 **to** n

$a_j \leftarrow H_{agg}((X_1, \ldots, X_n), X_j)$

if $g^{z_j} \neq R_{j,1} \cdot R_{j,2}^b \cdot X_j^{a_j \cdot c}$

return $\perp$

$z \leftarrow \sum_{i=1}^n z_i$

$R \leftarrow \prod_{i=1}^n R_i$

$\sigma \leftarrow (R, z)$

return σ

Verify(σ, Y, m)

parse $\sigma \leftarrow (R, z)$

$c \leftarrow H_1(R, Y, m)$

return 1 **if** $g^z = R \cdot Y^c$, **else** 0

Figure 6.2: *The MuSig2 scheme*

6.4 MuSig1 vs MuSig2

Scheme	Rounds	Communication (total)	Computation (per signer)
MuSig1	3	$n \cdot (n \cdot \mathbb{G} + (n+1) \cdot \mathbb{Z}_p)$	$(n+3) \cdot Hh + (n-1) \cdot GMul + GExp + 2 \cdot SMul$
MuSig2	2	$n \cdot (2 \cdot n \cdot \mathbb{G} + \mathbb{Z}_p)$	$3 \cdot Hh + 2 \cdot n \cdot GMul + 3 \cdot GExp + 3 \cdot SMul$

Figure 6.3: *Comparison of MuSig1 and MuSig2*

From the table we can see that MuSig2 is more expensive in both communication and computation complexity since $\mathbb{G}$ is larger than $\mathbb{Z}_p$ and GExp, GMul are the most expensive operations. However, the most important metric when it comes to multi- and threshold signatures is the amount of rounds. Here, MuSig2 definitely take the upper hand as it reduced the number of rounds by one while still allowing concurrent sessions.

Chapter 7

Distributed Key Generation for Threshold Signatures

Multi-signature and threshold signature schemes are fundamentally different in the way that they work. Multi-signature schemes are n-out-of-n meaning that all signers must contribute to the signature whereas threshold signature schemes are t-out-of-n meaning that t signers must contribute to the signature. An extremely important thing to note is that in the threshold setting, every subgroup containing t (honest) signers must be able to produce a signature which verifies under the public key of the entire group. This is important because that means that the public, private key pair of each signer is not so simple to create as different subsets of signers would then form different group keys, something which is undesirable. Instead, we need a way such that any t public keys of participants aggregate to the same public key for the group. To do so, we need to utilize t-out-of-n secret sharing schemes. The issue with that is that these schemes rely on a trusted dealer which is also something we do not want. Luckily, there exist many Distributed Key Generation (DKG) algorithms which can work without a trusted dealer. In this chapter, we describe such algorithms that are extensively in threshold signature schemes.

7.1 Secret Sharing Schemes

We describe some of the secret sharing schemes which are very similar to Shamir's Secret Sharing described in Section 3.1.7. The goal of these schemes is to distribute shares of a secret s to n members such that any group of at least t honest parties can recreate the key.

7.1.1 Shamir's Secret Sharing (SSS) [1]

The dealer samples $t-1$ coefficients $a_1, \cdots, a_{t-1}$ at random and defines a polynomial of degree $t-1$ using these values. He sets the constant term as the secret ($f(0) = s$):

$$f(x) = s + \sum_{i=1}^{t-1} a_i \cdot x^i \tag{7.1}$$

The secret share of each participant P_i is $(i, f(i))$. Using Lagrange interpolation, t participants can recreate the secret s.

Notes on Shamir's Secret Sharing:

- *The dealer is trusted, otherwise we have no guarantee that the shares which he sends to the participants are correct. If they are not correct, the secret s cannot be reconstructed.*
- *During interpolation, the members are trusted to send out the correct shares. If they do not, a different message s' will be recreated such that $s' \neq s$.*

7.1.2 Feldman's Verifiable Secret Sharing (VSS) [2]

In this secret sharing scheme, we do not trust the dealer. Therefore, we need to have some guarantee that the shares he distributes are correct. To do so, we build on SSS by adding an extra commitment step on the polynomial. This way each participant can check if their share is indeed correct i.e if it of the form $(i, f(i))$.
The first step is the same as in SSS; the dealer samples $t-1$ coefficients $a_1, \cdots, a_{t-1}$ at random and defines a polynomial of degree $t-1$ using these values. He sets the constant term as the secret ($f(0) = s$):

$$f(x) = s + \sum_{i=1}^{t-1} a_i \cdot x^i \tag{7.2}$$

When sending the private share $(i, f(i))$ to each participant P_i, the dealer also broadcasts a public commitment vector $\vec{C} = \langle \phi_0, \phi_1, \cdots, \phi_{t-1} \rangle$, where $\phi_0 = g^s$ and $\phi_j = g^{a_j}, j \in [t-1]$. Once a participant receives their share (i, v), they can check that is is correct i.e $v = f(i)$ by using the following equation:

$$g^v \stackrel{?}{=} \prod_{j=0}^{t-1} \phi_j^{i^j} = g^{f(i)} \tag{7.3}$$

Notes on Feldman's Verifiable Secret Sharing:

- *The participants must all have the same view of $\vec{C}$.*
- *Equation 7.3 can be used during reconstruction time to detect incorrect shares submitted by dishonest parties.*
- *By revealing $\phi_0 = g^s$, some information about the secret is leaked.*

7.1.3 Pedersen's Verifiable Secret Sharing (PVSS) [3]

Feldman's VSS reveals some information about the secret (since it reveals g^s). We would like a sharing scheme that would reveal no information. In order to do so, we use Pedersen's commitments.
The dealer samples coefficients for two polynomials: $a_1, \cdots, a_{t-1}$ for $f(x)$ and $a'_0, a'_1, \cdots, a'_{t-1}$ for $g(x)$ and sets $f(0) = s$:

$$f(x) = s + \sum_{i=1}^{t-1} a_i \cdot x^i \tag{7.4}$$

$$g(x) = \sum_{i=0}^{t-1} a'_i \cdot x^i \tag{7.5}$$

The secret share of participant P_i will now be $(i, f(i), g(i))$. While distributing these shares to each participant the dealer also broadcasts a public commitment vector $\vec{C} = \langle \phi_0, \cdots, \phi_{t-1} \rangle$ where $\phi_0 = g^s \cdot h^{a'_0}$ and $\phi_j = g^{a_j} \cdot h^{a'_j}$.
Once a participant receives their share (i, u, v) they can check that it is correct i.e $u = f(i)$ and $v = g(i)$ as follows:

$$g^u \cdot h^v \stackrel{?}{=} \prod_{j=0}^{t-1} \phi^{i^j} = g^{f(i)} \cdot h^{g(i)} \tag{7.6}$$

Notes on Pedersen's Verifiable Secret Sharing:

- *The value of ϕ_0 is now a Pedersen commitment on s which hides s unconditionally.*
- *It is shown in [3] that the dealer can succeed in distributing incorrect shares if he can solve $\log_g(h)$.*

7.2 Pedersen's DKG algorithm and PedPoP

One of the most used DKG algorithms is the well-known Pedersen's DKG [20]. In fact, a variant of Pedersen's DKG protocol appears in the state-of-the-art threshold signing protocol FROST [21] ; this new variant is called PedPoP (Pedersen + Proofs of Possession). In this section we describe the two schemes which are very similar.
We discuss briefly the intuition behind this DKG algorithm. In Pedersen's DKG, we run n parallel VSS algorithms with each one of our n participants being the dealer in a specific VSS session. This way, each participant will create a specific polynomial $\{f_i(x)\}_{i\in[n]}$. Each dealer, will distribute shares of his secret $f_i(0)$ to all n members. The final secret of the group will be $s = \sum_{j=0}^{n} f_j(0)$ and each participants P_i secret share will be $s_i = \sum_{j=0}^{n} f_j(i)$. Consider the following figure for an explanation:

$$\begin{aligned}
P_1 &: f_1(x) = a_{1,0} + a_{1,1} \cdot x + a_{1,2} \cdot x^2 + \cdots + a_{1,t-1} \cdot x^{t-1} \\
P_2 &: f_2(x) = a_{2,0} + a_{2,1} \cdot x + a_{2,2} \cdot x^2 + \cdots + a_{2,t-1} \cdot x^{t-1} \\
&\vdots \\
P_n &: f_n(x) = a_{n,0} + a_{n,1} \cdot x + a_{n,2} \cdot x^2 + \cdots + a_{n,t-1} \cdot x^{t-1} \\
\hline
F(x) &= s + A_1 \cdot x + A_2 \cdot x^2 + \cdots + A_{t-1} \cdot x^{t-1}
\end{aligned}$$

Figure 7.1: *Intuition behind Pedersen DKG*

In Figure 7.1 we can see the sampled $t-1$ polynomial of each participant acting as a dealer. If we were to sum these polynomials we would create an idealized polynomial $F(x) = \sum_{i=0}^{n} f_i(x)$ (this polynomial is idealized because it is never actually created during the DKG) and the shares of each participant will be $s_i = F(i) = \sum_{j=0}^{n} f_j(i)$.

Participants whose shares do not check out using Equation 7.3 get disqualified from the protocol and their shares will not be used. The way that they get disqualified is that if a share of P_j does not verify for P_i, then P_i will raise a complaint against P_j by publishing the share and the digital signature that accompanied it. The other participants will then see if the complaint is valid or not.

Pedersen DKG and PedPoP

Round 1

1. Every participant P_i samples t random values $(a_{i,0}, \ldots, a_{i,t-1}) \xleftarrow{\$} \mathbb{Z}_p$, and uses these values as coefficients to define a degree $t-1$ polynomial $f_i(x) = \sum_{j=0}^{t-1} a_j x^j$.
2. Every P_i computes a proof of knowledge to the corresponding secret $a_{i,0}$ by calculating $\sigma_i = (R_i, \mu_i)$, such that $k \xleftarrow{\$} \mathbb{Z}_p$, $R_i = g^k$, $c_i = \mathcal{H}(i, \Phi, g^{a_{i,0}}, R_i)$, $\mu_i = k + a_{i,0} \cdot c_i$, with Φ being a context string to prevent replay attacks.
3. Every participant P_i computes a public commitment $\vec{C}_i = (\phi_{i,0}, \ldots, \phi_{i,t-1})$, where $\phi_{i,j} = g^{a_{i,j}}, 0 \leq j \leq t-1$.
4. Every P_i broadcasts $\vec{C}_i$, σ_i to all other participants.
5. Upon receiving $\vec{C}_\ell, \sigma_\ell$ from participants $1 \leq \ell \leq n, \ell \neq i$, participant P_i verifies $\sigma_\ell = (R_\ell, \mu_\ell)$, making a complaint against P_j on failure, by checking $R_\ell \stackrel{?}{=} g^{\mu_\ell} \cdot \phi_{\ell,0}^{-c_\ell}$, where $c_\ell = \mathcal{H}(\ell, \Phi, \phi_{\ell,0}, R_\ell)$.

   Upon success, participants delete $\{\sigma_\ell : 1 \leq \ell \leq n\}$.

Round 2

1. Each P_i securely sends to each other participant P_ℓ a secret share $(\ell, f_i(\ell))$, deleting f_i and each share afterward except for $(i, f_i(i))$, which they keep for themselves.
2. Each P_i verifies their shares by calculating: $g^{f_\ell(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{\ell,k}^{i^k}$, raising a complaint if the check fails.
3. Let $QUAL$ be the set of participants that haven't been disqualified from complaints. Each P_i calculates their long-lived private signing share by computing $s_i = \sum_{\ell \in QUAL} f_\ell(i)$, stores s_i securely, and deletes each $f_\ell(i)$.
4. Each P_i calculates their public verification share $X_i = g^{s_i}$, and the group's public key $\tilde{X} = \prod_{j \in QUAL} \phi_{j,0}$. Any participant can compute the public verification share of any other participant by calculating $X_i = \prod_{j \in [n]} \prod_{k=0}^{t-1} \phi_{j,k}^{i^k}$.

Figure 7.2: *Pedersen DKG and PedPoP. The pink part is only present in PedPoP*

In the Figure 7.2 we can see the steps of Pedersen's DKG in detail and the difference between Pedersen's DKG and PedPoP where a proof-of-possession of the constant term is

included to avoid rogue-key attacks [47] , [49].

Overall, Pedersen DKG is a simple two rounds algorithm making it very appealing. However, it came with no security proof as it was not known what properties a DKG algorithm should have. In the next section, we discuss an influence attack on Pedersen DKG and how Gennaro et al. [22] fixed the issue.

7.3 Gennaro's DKG algorithm

7.3.1 Problems with Pedersen's DKG

Without specific security goals in mind it was impossible for Pedersen DKG to set a standard on what properties a DKG ought to have. In their paper, [22] Gennaro et al, constructed an influence attack against Pedersen DKG and after defining some specific properties for DKG algorithms, they created their own version which avoided said attack. This influence attack does not break Pedersen DKG per se, in fact in the same paper it is proven that Pedersen DKG is "secure enough" when it comes to specific applications like Schnorr signature (in fact it is secure in all applications where the security of the underlying protocol can be directly reduced to the hardness of the discrete logarithm problem). The fact that specific schemes like FROST [21] use versions of Pedersen DKG is a clear indication that it is trusted. Nonetheless, Gennaro et al. introduced some properties that a DKG algorithm should have. After defining those, they were able to mount an attack against Pedersen DKG that breaks one of those properties. The properties defined are:

- *Correctness*
  - *All subsets of t shares provided by honest parties define the same unique secret key sk.*
  - *The resulting secret key sk is uniformly distributed (and hence the associated public key also follows the uniform distribution).*
- *Secrecy*
  - *No information on sk can be learned by an adversary except for what is already implied by the public key (in discrete logarithm systems, that is g^{sk}).*

The influence attack we are about to describe breaks the second bullet point of Correctness, meaning that an adversary can influence the distribution of the resulting key so that the output does not follow the uniform distribution. The basic idea of the attack is that an adversary participating in the Pedersen DKG can make complaints against participants after seeing their $\phi_{i,0}$ values (if the complaint stands then that member will not be included in QUAL). This is important as the final public key of the group is calculated as $\tilde{X} = \prod_{i \in QUAL} \phi_{i,0}$. The attack works as follows: Assume the adversary controls two parties P_1, P_2 and he wants to bias the output of the DKG so that the resulting public key has 0 as its last bit. At the end of the first round of the DKG (after having access to the commitment vectors of the participants) the adversary can calculate three possible keys (assuming the rest of the participants are honest and will thus be in QUAL until the end):

- *Public key if both P_1 and P_2 are in QUAL : $pk_{hon} = \prod_{i=1}^{n} \phi_{i,0}$.*
- *Public key if P_1 is disqualified from QUAL: $pk_{P_1} = \prod_{i \neq P_1}^{n} \phi_{i,0}$.*
- *Public key if P_2 is disqualified from QUAL: $pk_{P_2} = \prod_{i \neq P_2}^{n} \phi_{i,0}$.*

If pk_{hon} ends in 0 then the adversary does not have to do anything and can just follow the DKG like any honest member. Otherwise, it is quite possible that one of pk_{P_1} or pk_{P_2} ends in 0. If pk_{P_1} (or pk_{P_2}) ends in 0 then the adversary can have P_2 (or P_1) make a complaint against P_1 (or P_2) during step 2 of the second round (since he controls both parties, it is easy for him to create a complaint that will stand i.e VSS verification will fail). This was the adversary can influence the final output effectively breaking the property that requires the resulting keys follow the uniform distribution.

7.3.2 Fixing the issues in Pedersen's DKG

Gennaro el al. did not just create this influence attack against Pedersen DKG ; they also created a scheme to combat the issue and which provably has the properties described in the previous section. As we will see however, there is a tradeoff between the security of Pedersen DKG and the increased cost of the new DKG. We next describe the new DKG and discuss the tradeoffs between the two algorithms. In this new DKG algorithm, we will have n parallel runs of PVSS (instead of VSS) with each of the n participants being a dealer in a specific PVSS session. These sessions will hide the public key before the algorithm is concluded. The reason for that is that the new commitments will be $\phi_{i,0} = g^{a_{i,0}} \cdot h^{a'_{i,0}}$ however the public key will still be $\tilde{X} = \prod_{i \in QUAL} g^{a_{i,0}}$. This way the adversary will not be able to find the possible public keys since they are blinded. The new DKG is described in Figure 7.3. Some note on the new DKG and the tradeoffs between this algorithm and Pedersen DKG:

- *The set QUAL is built during PVSS. All parties included in QUAL will influence the final key.*
- *If a party misbehaves during VSS (round 3, step 2) the honest parties can recover his polynomial and his contribution to the secret key will still be included (otherwise the adversary would be able to use the attack as in Pedersen DKG).*
- *The new protocol adds two rounds (one if no party is dishonest) to Pedersen DKG and demands at most twice more local computation (VSS + PVSS).*

7.4 A formal analysis of DKG algorithms

Building upon existing notions, Komlo et al. [23] wanted to define game-based notions of security for DKG algorithms because previous notions were either incomplete or informally presented. Apart from that, they presented a generic construction of a secure DKG using three modular cryptographic building blocks which have their own properties. Using this new construction, in order to create a new DKG protocol, it is just needed to use building

Gennaro DKG

Round 1

1. Every participant P_i samples t random values $(a_{i,0}, \ldots, a_{i,t-1}) \xleftarrow{\$} \mathbb{Z}_p$, and uses these values as coefficients to define a degree $t-1$ polynomial $f_i(x) = \sum_{j=0}^{t-1} a_j x^j$.

2. Every P_i broadcasts $\vec{C}_i = \langle \phi_{i,0}, \cdots, \phi_{i,t-1} \rangle$, to all other participants where $\phi_{i,j} = g^{a_{i,j}} \cdot h^{a'_{i,j}}$.

Round 2

1. After receiving all commitment vectors, each participant P_i secretly sends $s_{i,j} = f_i(j)$ and $s'_{i,j} = g_i(j)$ to all other participants $P_j, j \neq i$ and keeps $s_{i,i} = f_i(i)$ and $s'_{i,i} = g_i(i)$ for himself.

2. Each P_i verifies the validity of the secret shares $(s_{j,i}, s'_{j,i})$ sent to him from other participants $P_j, j \neq i$ broadcasting a complaint against P_j if the shares are not valid. If the complaint is valid, then P_j is disqualified.

3. Let $QUAL$ be the set of participants that haven't been disqualified from complaints. Each P_i calculates their long-lived private signing share by computing $s_i = \sum_{\ell \in QUAL} f_\ell(i)$, stores s_i securely, and deletes each $f_\ell(i)$. The secret key of the group can be found via Lagrange interpolation.

Round 3

1. Each party $P_i, i \in QUAL$ broadcasts $\vec{A}_i = \langle g^{a_{i,0}}, \cdot, g^{a_{i,t-1}} \rangle$.

2. Each participant P_i verifies the shares $s_{j,i}$ via Feldman's VSS. If the check fails he broadcasts a complaint against P_j displaying the value $s_{j,i}, s'_{j,i}$ which passed validation during the previous round but failed on this step.

3. For parties P_i that receive at least one valid complaint, the other parties run the reconstruction to compute $a_{i,0}, f_i(z)$ in the clear. The public key is calculated as $\tilde{X} = \prod_{j \in QUAL} A_{j,0}$.

Figure 7.3: *DKG by Gennaro et al.*

blocks that follow specific properties ; the security of the DKG is then proven for free as long as its building blocks do not break any of the required characteristics. In this section we briefly describe the building blocks of a DKG construction and the properties they should hold. Finally, we discuss the properties of a secure DKG and give the construction using generic building blocks. For proofs and game definitions of properties we advise reading [23].

7.4.1 Building blocks of a secure DKG

The three building blocks for a DKG are:

- *A secure hash function, we talk about hash function in Section 3.1.2.*
- *An Aggregatable Verifiable Secret Sharing (AgVSS) scheme: Building on a VSS, an AgVSS allows us to aggregate multiple runs of a VSS protocol into a single instance. Apart from that, it adds a small modification called a tweak to the original shares so that the resulting public information is not visible before has ended. An AgVSS consists of the tuple of algorithms (Share, Verify, Recover, GetPub, GetTweak, AggPriv, AggPub):*
  - *Share*$(\kappa, s, n, t) \rightarrow (\{1, w_1), \cdots, (n, w_n)\}, D$*: Where* $\{i, w_i\}_{i\in[n]}$ *are the shares and* D *is a commitment on those shares.*
  - *Verify*$(i, w_i, D) \rightarrow \{0, 1\}$
  - *Recover*$(t, M) \rightarrow s/\perp$*: Where* $M = \{(j, w_j)\}_{j\in[t]}$.
  - *GetPub*$(i, D) \rightarrow W_i$*: Where* W_i *is the public value associated with the secret value* w_i.
  - *GetTweak*$(O, aux) \rightarrow v$*: Where* $O = \{D_1, \cdots, D_\ell\}$.
  - *AggPriv*$(P, v) \rightarrow \hat{w_i}$*: Where* $P = \{w_{j,i}\}_{j\in[\ell]}$.
  - *AggPub*$(O, v) \rightarrow C$*: Where* C *is the aggregated commitment.*

  Using VSS as an example (ignoring the tweak for now), this VSS can be aggregated as:
  $P_1 : f_1(z) = a_{1,0} + a_{1,1} \cdot z + \cdots + a_{1,t-1} \cdot z^{t-1}$ *with* $D_1 = \{g^{a_{1,0}}, g^{a_{1,1}}, \cdots, g^{a_{1,t-1}}\}$.
  $\cdots$
  $\cdots$
  $P_n : f_n(z) = a_{n,0} + a_{n,1} \cdot z + \cdots + a_{n,t-1} \cdot z^{t-1}$ *with* $D_n = \{g^{a_{n,0}}, g^{a_{n,1}}, \cdots, g^{a_{n,t-1}}\}$.
  These runs can be aggregated into a single run with a single commitment:
  $F(z) = A_0 + A_1 \cdot z + \cdots + A_{t-1} \cdot z^{t-1}$ *with* $C = \{\hat{A_0}, \hat{A_1}, \cdots, \hat{A_{t-1}}\}$ *where* $A_i = \sum_{j=1}^{n} a_{j,i}$ *and* $\hat{A_i} = \prod_{j=1}^{n} g^{a_{j,i}}$ *and the new secret shares of each participant* P_i *being* $\hat{w_i} = \sum_{j=1}^{n} w_{j,i}$.

  Now we can add the tweak value $v = H(\{D_1, \cdots, D_n\}, aux)$ *which will turn the secret shares of the participants to* $\hat{w_i} = v + \sum_{j=1}^{n} w_{j,i}$ *and the aggregated commitment to* $C = \{g^v \cdot \hat{A_0}, \hat{A_1}, \cdots, \hat{A_{t-1}}\}$.

In order for an AgVSS to be used in the DKG construction it must satisfy the properties of aggregated correctness, aggregated secrecy and uniqueness.

- *Aggregated correctness : For honestly generated shares, tweak the aggregated shares must verify with the aggregated commitment and after using Recover the correct secret must be returned.*
- *Aggregated secrecy : The adversary cannot distinguish between honest shares and simulated shares.*
- *Uniqueness : The adversary cannot find two different recovery sets for the same commitment but which recover distinct secrets.*

- *A Non-Interactive Key Exchange (NIKE) scheme: A NIKE is a cryptographic primitive which enables two parties, who know each others' public keys, to agree on a symmetric shared key without requiring any interaction. It is a tuple of the algorithms (KeyGen, Verify, SharedKey):*
  - $KeyGen(\kappa) \rightarrow (sk, pk)$
  - $Verify(sk, pk) \rightarrow \{0, 1\}$
  - $SharedKey(sk_1, pk_2) \rightarrow \psi$: *Where* ψ *is the shared key.*

  The canonical example of a NIKE scheme is the Diffie-Hellman key exchange.

  In order for a NIKE to be used in the DKG construction, it must satisfy the properties of correctness, session-key unrecoverability and bindness.
  - *Correctness : A NIKE is correct if every key pair generated by KeyGen passes verification by Verify and for every honestly generated key pairs,* $SharedKey(sk_1, pk_2) = SharedKey(sk_2, pk_1)$.
  - *Session-key unrecoverability : A NIKE is session-key unrecoverable if an adversary, given access to any two honestly generated public keys, cannot obtain the shared secret.*
  - *Bindness : A NIKE is binding if for key pairs that pass Verify it holds that* $SharedKey(sk_1, pk_2) = SharedKey(sk_2, pk_1)$.

7.4.2 Properties of a secure DKG and construction from generic building blocks

A secure DKG must satisfy the properties of correctness, strong/weak robustness, zero-knowledge and indistinguishability:

- *Correctness : Honestly generated key shares must recover the secret key which verifies with the public key on the target key generation scheme.*
- *- Robustness:*
  - *Strong : The protocol should always succeed as long as at least t honest parties are present.*

- *Weak : The protocol can fail as long as all honest parties agree that it failed i.e they all have the same status after the end of the protocol.*

- *Zero-Knowledge : The adversary does not gain any additional advantage to learn sk than it would against the target key generation algorithm. This is once again shown via a simulation algorithm just like the aggregated secrecy property of the AgVSS scheme.*

- *Indistinguishability : The DKG must generate keys that are indistinguishable from keys output by the target key generation algorithm.*

The reader is encouraged to consult [23] for a more detailed description of the properties as well as their game-based definitions. In Figure 7.4 we describe the generic DKG initiated by a NIKE NK*, an AgVSS* AV *and a hash function* H*. Without getting into too many details, we can say that this generic DKG construction is secure since no one can learn the tweak until the Finalize round because in order to learn the tweak one must know the shared keys (which are revealed at the end of the second round) or the* a_j *values which get reconstructed only if* j *is a corrupt member. Finally, even if* j *is corrupt, his value* a_j *will be used in the final output key pair of the DKG.*

PerformRound$_0$[H](κ, n, t, i)

$(a_i, A_i) \xleftarrow{\$} \mathsf{NK.KeyGen}(\kappa)$

$\text{in} \leftarrow (\kappa, a_i, n, t)$

$((\{w_{i,j}\}_{j\neq i}), D_i) \leftarrow \mathsf{AVH}[i].\mathsf{Share}(\text{in})$

$(\beta_i, B_i) \leftarrow \mathsf{NK.KeyGen}(\kappa)$

$\text{bmsg}_{0,i} \leftarrow (A_i, B_i, D_i)$

for $j \in [n], j \neq i$

$\text{pmsg}_{0,ij} \leftarrow (w_{i,j})$

$\text{state}_i \leftarrow (\beta_i, w_{ii}, D_i, B_i)$

$\text{out}_p \leftarrow \{\text{pmsg}_{0,ij}\}_{i\in[n],j\neq i}$

return $(\text{state}_i, \text{out}_p, \text{bmsg}_{0,1})$

PerformRound$_1$[H]$(\text{state}_i, \text{inp}_i, \text{inb}_i)$

parse $\{(i, w_{ji})\}_{j\in[n],j\neq i} \leftarrow \text{inp}_1$

parse $((A_j, B_j, D_j))_{j\in[n],j\neq i} \leftarrow \text{inb}_1$

for $j \in [n], j \neq i$ **do**

if $\mathsf{AV(H1).Verify}(i, w_{ji}, D_j) \neq A_j$

or if $\mathsf{AV(H1).GetPub}(0, D_j) \neq A_j$

$\text{state}_i.\text{status} \leftarrow \text{abort}$

return $(\text{state}_i, \perp, \text{bmsg}_{1,i}, \perp)$

$\text{state}_i. \leftarrow \text{state}_i \cup \{(j, w_{ji})\}_{j\in[n]}$

$\text{bmsg}_{1,i} \leftarrow \text{accept}$

return $(\text{state}_i, \perp, \text{bmsg}_{1,i})$

PerformRound$_2$[H]$(\text{state}_i, \emptyset, \text{inb}_2)$

if $\text{fail} \in \text{inb}_2$

$\text{state}_i.\text{status} \leftarrow \text{abort}$

return $(\text{state}_i, \perp, \perp)$

$\text{bmsg}_{2,i} \leftarrow \beta_i$

return $(\text{state}_i, \perp, \text{bmsg}_{2,i})$

Finalize[H]$(\text{state}_i, \emptyset, \text{inb}_3)$

if $\text{state}_i.\text{status} = \text{abort}$

return $(\perp, \perp, \perp), (\text{abort}, \perp)$

parse $\{\beta_j\}_{j\in[n],j\neq i} \leftarrow \text{inb}_3$

$\text{qual} \leftarrow \emptyset; \text{corrupt} \leftarrow \emptyset$

for$j \in [n], j \neq i$ **do**

if $\mathsf{NK.Verify}(\beta_j, B_j) = 1$

$\psi_j \leftarrow \mathsf{NK.SharedKey}(\beta_j, A_j)$

$\text{qual} \leftarrow \text{qual} \cup \{j\}$

else $\text{corrupt} \leftarrow \text{corrupt} \cup \{j\}$

for $j \in \text{corrupt}$ **do**

$M_j = \{(k, w_{jk})\}_{k\in qual}$

$a_j \leftarrow \mathsf{AV[H1].Recover}(t, M_j)$

$\psi_j \leftarrow \mathsf{NK.SharedKey}(a_j, B_j)$

$\upsilon \leftarrow$
$\mathsf{AV[H1].GetTweak}(\{D_j\}_{j\in[n]}, \{\psi_j\}_{j\in[n]})$

$sk_i \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{AggPriv}(i, \{w_{ji}\}_{j\in[n]}, \upsilon)$

$C \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{AggPub}(\{D_j\}_{j\in[n]}, \upsilon)$

$pk \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(0, C)$

for $i \in \text{qual}$ **do**

$pk_i \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(i, C)$

$aux \leftarrow \{pk_i\}_{i\in\text{qual}}$

return $(pk, \text{qual}, aux), (\text{accept}, sk_i)$

Figure 7.4: *Generic DKG construction*

Chapter 8

The FROST and FROST2 Threshold Schemes

Now that we have discussed key generation in threshold signatures, we can look at some threshold signing schemes, specifically the FROST schemes [21] , [24]. The FROST scheme [21] was one of the first concurrently secure threshold Schnorr schemes which allowed two-round signing without using pairings. FROST stands for: Flexible Round-Optimized Schnorr Threshold Signatures.

- *Flexible: As we will see later, FROST is a two-round scheme where the first round can be precomputed to cover multiple signing sessions. This allows the signers to perform signing operations asynchronously ; once the first round (also called the pre-processing round) is complete, signers only need to eventually reply with a single message for the group signature to be completed.*
- *Round-Optimized: Can be used as either a two-round protocol, or optimized to a single-round protocol with a pre-processing stage.*
- *Schnorr: The resulting group signature is identical to a normal Schnorr that can be verified using a single aggregated group key.*
- *Threshold: A threshold t out of n possible participants are required for signing operations.*

Each FROST signing session will contain t participants all of which are expected to send partial signatures in order for a group signature to be formed. If even a single signing member does not reply with a partial signature, then the signing session cannot be completed. Likewise, if a member replies with a partial signature that does not verify under his public key, the signing session cannot be completed either. In both such cases, we would need to create a new session without including the delaying party (the party which does not send out signatures after a timer) or the party whose partial signature share does not verify. Therefore, FROST is what is referred to as a non-robust scheme meaning that if a party misbehaves, the protocol needs to be restarted after kicking out the misbehaving participant. Of course, this is not new ; both MuSig1 and MuSig2 schemes presented in previous chapters where also not robust. The difference between MuSig and FROST is that the former is a multi-signature scheme whereas the latter is a threshold signing scheme. In MuSig we cannot kick out participants since all n of them are expected to contribute

to the final signature, whereas in FROST, even if we kick out misbehaving members, the threshold t might still be covered by the remaining parties. FROST traded off the robustness aspect for efficiency, robust threshold signing schemes such as [50] required more than two communication rounds (which is what FROST requires). This combined, with the fact that the first round of FROST can be precomputed means than in the case of a misbehaving participant, essentially only a single extra round will need to take place. Nonetheless, we describe how to add robustness to FROST in Section 8.3. We now describe the algorithms of FROST and FROST2 which are very similar to the ones in MuSig2.

8.1 Algorithms of FROST/FROST2

- *Setup. We work in a group $\mathbb{G}$ of order p and g is a generator of the group. The scheme will also need two hash functions $H_{sig}, H_{non} \rightarrow \mathbb{Z}_p$.*

- *KeyGen. We need to use a DKG algorithm like PedPoP with n signers and threshold t. At the end of the protocol, each signer holds a secret key x_i, a public key X_i and the group has an aggregate key $\tilde{X}$.*

- *$Sign_1$: We assume the signers wish to sign an (agreed-on) message m. We also assume that the signing set SS of the session is known to the participants. Each signer P_i generates two random nonces $d_i, e_i \xleftarrow{\$} \mathbb{Z}_p$, computes $D_i = g^{d_i}, E_i = g^{e_i}$ and sends out D_i, E_i to all other signers.*

- *$Sign_2$: After receiving all $\{(D_j, E_j)\}_{j \in SS}$ from all other participants, participant P_i computes the binding factors $b_j = H_{non}(j, \tilde{X}, m, \{(\ell, D_\ell, E_\ell)\}_{\ell \in SS}), j \in SS$ (or the single binding factor $b = H_{non}(\tilde{X}, m, \{(\ell, D_\ell, E_\ell)\}_{\ell \in SS}$ in the case of FROST2), the group commitment $R = \prod_{j \in SS} D_j \cdot E_j^{b_j}$ or ($R = \prod_{j \in SS} D_j \cdot E_j^{b}$ in the case of FROST2), the challenge of the Schnorr signature $c = H_{sig}(\tilde{X}, R, m)$, his Lagrange coefficient λ_i for the set SS and his partial signature share $z_i = d_i + e_i \cdot b + c \cdot \lambda_i \cdot x_i$ which he sends to the coordinator.*

- *Combine. After all partial signatures $\{z_i\}$ have been sent out and verified, the final group signature on m is produced as $\sigma = (R, \sum_{i \in SS} z_i)$.*

- *Verify. Any public validator can verify the signature under the public key $\tilde{X}$ by using the Schnorr single party verification algorithm on σ.*

  In Figure 8.1 we can see the algorithms of the 2 schemes in detail. The authors of FROST suggest pre-processing the first round such that every participant sends out multiple tuples (D_{ij}, E_{ij}) to the coordinator. This way when a participant gets added to a session, the first round of the protocol can be skipped as nonces will be available. Of course, this is not specific to the FROST protocol ; signers in MuSig2 can do the exact same thing and likewise signers of MuSig1 can pre-process the first round by sending out multiple hashes. In order for rounds to be pre-processed, a coordinator must be present ; this is fine for our analysis as we always assume our protocols have a coordinator.

Setup(1^κ)

$(G, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

$H_{non}, H_{sig} : \{0,1\}^* \to \mathbb{Z}_p$

$par \leftarrow ((\mathbb{G}, p, g), H_{non}, H_{sig})$

return par

KeyGen(t, n, par)

$(\tilde{X}, \{(X_j, x_j\}_{j\in[n]}) \leftarrow \mathrm{PedPoP}(t, n)$

for j = 1 **to** n:

$pk_j \leftarrow X_j$

$sk_j \leftarrow x_j$

return $\tilde{X}, \{pk_j\}_{j\in[n]}, \{sk_j\}_{j\in[n]}$

Verify(σ, Y, m)

parse $\sigma \leftarrow (R, z)$

$c \leftarrow H_{sig}(R, Y, m)$

return 1 **if** $g^z = R \cdot Y^c$, **else** 0

Sign$_1(SS, m, i, \{pm_{0,j}\}_{j\in SS}, sk_i, st_i)$

$d_i, r_i \leftarrow \mathbb{Z}_p$

$D_i \leftarrow g^{d_i}$

$E_i \leftarrow g^{e_i}$

$pm_{1,i} \leftarrow (D_i, E_i)$

$st_i \leftarrow (d_i, r_i)$

return $pm_{1,i}, st_i$

Sign$_2(SS, m, i, \{pm_{1,j}\}_{j\in SS}, sk_i, st_i)$,

parse $(d_i, e_i) \leftarrow st_i, x_i \leftarrow sk_i$

parse $\{(D_j, E_j)\}_{j\in SS} \leftarrow \{pm_{1,j}\}_{j\in SS}$

$b \leftarrow H_{non}(\tilde{X}, m, \{(\ell, D_\ell, E_\ell)\}_{\ell\in SS}$

$R \leftarrow \prod_{j\in SS} D_j \cdot E_j^b$

for j ∈ SS

$b_j \leftarrow H_{non}(j, \tilde{X}, m, \{(\ell, D_\ell, E_\ell)\}_{\ell\in SS}$

$R \leftarrow \prod_{j\in SS} D_j \cdot E_j^{b_j}$

$c \leftarrow H_{sig}(R, \tilde{X}, m)$

$z_i \leftarrow d_i + e_i \cdot b_i + c \cdot \lambda_i \cdot x_i$

$pm_{2,i} \leftarrow z_i$

return $pm_{2,i}, st_i$

Combine$(\{pm_{k,j}\}_{k\in[2],j\in SS})$

parse $\{(D_j, E_j)\}_{j\in SS} \leftarrow \{pm_{1,j}\}_{j\in SS}$

parse $\{z_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

$b \leftarrow H_{non}(\tilde{X}, m, \{(\ell, D_\ell, E_\ell)\}_{\ell\in SS}$

$R \leftarrow \prod_{j\in SS} D_j \cdot E_j^b$

for j ∈ SS

$b_j \leftarrow H_{non}(j, \tilde{X}, m, \{(\ell, D_\ell, E_\ell)\}_{\ell\in SS}$

$R \leftarrow \prod_{j\in SS} D_j \cdot E_j^{b_j}$

$c \leftarrow H_{sig}(R, \tilde{X}, m)$

for j ∈ SS

if $g^{z_j} \neq D_j \cdot E_j^{b_j} \cdot X_j^{\lambda_j \cdot c}$

return $\perp$

$z \leftarrow \sum_{i\in SS} z_i$

$R \leftarrow \prod_{i\in SS} R_i$

$\sigma \leftarrow (R, z)$

return σ

Figure 8.1: *The FROST and FROST2 schemes. The parts in pink only appear in FROST2, while the parts in gray only appear in FROST*

8.2 Complexity analysis of FROST and FROST2

The communication complexity of the two schemes is the same:

Each session consists of 2 rounds. For each session, each signer has to send out elements of size:

- ***Sign***$_1$*:* $t \cdot 2 \cdot \mathbb{G}$ *(broadcast)*
- ***Sign***$_2$*:* $\mathbb{Z}_p$ *(sent to coordinator)*

Thus, the total amount of communication for a FROST/FROST2 session is: $t \cdot (t \cdot 2 \cdot \mathbb{G} + \mathbb{Z}_p)$

For each FROST session, each signer has to perform computations of cost:

- ***Sign***$_1$*:* $2 \cdot GExp$
- ***Sign***$_2$*:* $(t+1) \cdot Hh + 2 \cdot t \cdot GMul + t \cdot GExp + 3 \cdot SMul + Lagr$

Therefore, the total amount of computation per signer is $(t+1) \cdot Hh + 2 \cdot t \cdot GMul + (t+2) \cdot GExp + 3 \cdot SMul + Lagr$.

For each FROST2 session, each signer has to perform computations of cost:

- ***Sign***$_1$*:* $2 \cdot GExp$
- ***Sign***$_2$*:* $2 \cdot Hh + 2 \cdot t \cdot GMul + GExp + 3 \cdot SMul + Lagr$

Therefore, the total amount of computation per signer is $2 \cdot Hh + 2 \cdot t \cdot GMul + 3 \cdot GExp + 3 \cdot SMul + Lagr$.

Scheme	Rounds	Communication (total)	Computation (per signer)
FROST	2	$t \cdot (t \cdot 2 \cdot \mathbb{G} + \mathbb{Z}_p)$	$(t+1) \cdot Hh + 2 \cdot t \cdot GMul + (t+2) \cdot GExp + 3 \cdot SMul + Lagr$
FROST2	2	$t \cdot (t \cdot 2 \cdot \mathbb{G} + \mathbb{Z}_p)$	$2 \cdot Hh + 2 \cdot t \cdot GMul + 3 \cdot GExp + 3 \cdot SMul + Lagr$

Figure 8.2: *Comparison of FROST and FROST2*

Given that GExp is the most costly operation, we can clearly see the reduction in complexity costs that FROST2 was able to achieve over the original FROST scheme in Figure 8.2.

8.3 Turning FROST robust with ROAST

8.3.1 Robustness

As we mentioned before, the FROST schemes do not have the property of robustness. That means that if a party misbehaves by sending out a partial signature share which does not verify, the protocol needs to restart. Even more importantly, an adversary in a FROST session can simply choose to never reply with a partial signature. One might say that the solution to this is trivial : just abort the protocol if a specific time frame has passed and restart a new session. This however is a synchronous approach which assumes that protocol messages will reach their destination in a constant time for all signers. However, in

real-world applications over the internet it is much more desirable to create a robust scheme in the asynchronous model. In this model, our only guarantee is that protocol messages will reach their destination eventually. For this reason, when working in the asynchronous model, we cannot raise timers when messages are delayed as we run the risk of kicking out an honest party. Let's start by defining robustness and see why that would be catastrophic.

Definition 8.17. *Robustness in Threshold Signatures*
We call a threshold signing protocol (run with n signers) robust if it is guaranteed to output a valid signature in the presence of t honest signers, even if the remaining signers try to prevent the protocol from completing.

It is obvious from the previous definition that if we have exactly t honest signers, and we kick out one of them, then the remaining $t-1$ honest signers will not be able to produce a signature (due to the threshold) so we instantly lose robustness. Therefore, the synchronous approach will not work in the asynchronous model if robustness is our goal. Therefore achieving robustness is not a trivial task but is it necessary for many real-world applications. For example, if we wish to apply threshold signatures to DNS where DoS attacks are costly, robust schemes are needed as seen in [51]. Apart from that, we can list plenty of real world systems such as [52] [53] that use robust BLS signatures. If we wish for Schnorr signatures to be an alternative, we need ways to add robustness to Schnorr schemes.
In this chapter, we look at a wrapper protocol for FROST called ROAST [25] which adds robustness to the schemes. However, before we describe the scheme, we must first look at a specific property that robust schemes must have: Identifiable Abort.

8.3.2 Identifiable Abort property

Intuitively speaking, the Identifiable Abort property states that an adversary that misbehaves during a signing protocol will be identified (and then kicked out of the protocol). In order to do so, a scheme that has this property must have an extra ShareVal algorithm which can identify if a partial signature share is valid or not. In the identifiable abort game defined in [25], the adversary wins if the malicious signers under its control submit signature shares which pass validation (ShareVal) but lead to the output of an invalid group signature (break of accountability) or if an honest party outputs a share that does not pass validation (break of non-frameability).
It is easy to see that the FROST schemes have this property as each share is validated during the Combine algorithm as each share is a mini-Schnorr signature. This way break of non-freameability can never happen as the adversary does not control the information included in the partial signatures of honest signers, and break of accountability can also never occur as signature shares that pass the validation will create a correct signature due to Lagrange interpolation.

8.3.3 The ROAST wrapper protocol

We give an intuitive explanation of the ROAST protocol and refer the reader to [25] for a much more detailed description. ROAST [25] is a wrapper protocol on the FROST protocol

that guarantees robustness in the asynchronous model. The way that this protocol operates is that multiple parallel FROST sessions are run and within a specific number of sessions, at least one them is guaranteed to succeed. In order to understand the ROAST protocol in more detail, let's consider a specific run of a FROST session:

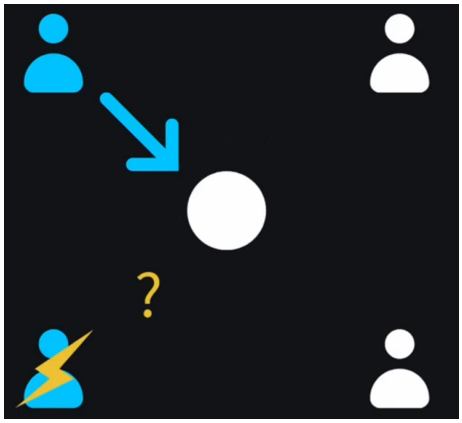

Figure 8.3: *FROST session in a 2-out-of-4 setting which includes the top and bottom left servers. The server on the top left has sent his partial signature but the bottom left signer is not responsive.*

We can make some observations on the session that appears in Figure 8.3:

- *There is no need to abort this session, as the unresponsive signer might reply.*
- *Unresponsive signers should be excluded from future sessions.*
- *We need to start a new session to guarantee progress.*
- *If we had piggybacked a new presignature share (that is a new nonce (D, E)) on every partial signature share, we could start a new session immediately.*

Based on these observation we can create a plan for the wrapper protocol:

- *We will keep sessions open, maybe pending signers will respond.*
- *We will maintain a set of responsive (not pending) nodes, these members can be used for future sessions.*
- *Whenever we have t responsive members in the previous set we will start a new FROST session with them.*
- *Each participant has to piggyback a fresh presignature share on every partial signature they send.*

With these things in mind, we can create the following ROAST algorithm that is event-driven:

ALGORITHM 8.1: *ROAST Protocol*

Signer Behavior:
Upon init:
    Send initial presignature share.
Upon receiving a presignature share (start of new session):
    Send partial signature.
    Send new presignature share.

Coordinator Behavior:
Upon init:
    Mark all nodes as UNRESPONSIVE.
Upon receiving an initial presignature share:
    Mark sender as RESPONSIVE.
Upon receiving a partial signature and new presignature share for session *sid*:
if partial signature does not verify **then**
        Mark sender as MALICIOUS and exclude them from future sessions.
end if
if session *sid* has t signature shares **then**
        Compute and output group signature.
end if
Mark sender as RESPONSIVE.
if there are t RESPONSIVE members **then**
        Start a new session with them.
        Send them their presignatures.
        Mark them as UNRESPONSIVE.
end if

In this wrapper protocol, the n members are split into two distinct group, the Responsive and the Unresponsive sets:

- *Responsive: A signer is in this set once a presignature is available for him (meaning he can be added in a new session). This can be done in two ways: either he was added in a session and he responded with his partial signature share and a new presignature or he has never been added to a session but has sent his initial presignature share.*
- *Unresponsive: A signer is in this set if he is blocking a session i.e he is not sending out his partial signature share.*

In order to justify why this protocol guarantees robustness we look at its invariant which states that any signer is Unresponsive in at most one FROST session. This is easy to see as Unresponsive signers will not be added to further sessions. Therefore each Unresponsive member can hold up at most 1 session. Furthermore, due to the Identifiable Abort property of FROST, a malicious signer cannot escape the Unresponsive state as he will either send out a non-valid partial signature (and be caught as malicious getting kicked out of future sessions) or he will send out a valid partial signature (in which case he does not act as an adversary and his signature share can be used to compute the final group signature). Therefore each adversarial signer can hold up to a single session (by staying Unresponsive forever). In order for the protocol to succeed we of course need at least t honest parties, this

means that the adversaries can be $f \leq n - t$. In the worst case, each malicious signer will be able to block a separate signing session. This means that if $f + 1$ sessions are started, it is guaranteed that one of them will succeed. Therefore, robustness is guaranteed with $n - t + 1$ sessions since in one of those sessions there will only be honest participants ; that session is guaranteed to end eventually.

Chapter 9

Security Notions for Threshold Signatures

In Section 4.3.2 we talked briefly about unforgeability for multi- and threshold signatures. Intuitively, the most basic definition of unforgeability for threshold signing schemes states that an adversary who control $t-1$ signers is unable to forge a signature. When it comes to FROST, its unforgeability proof is presented in [54]. However, this notion of unforgeability, although very natural, is not the only definition that exists in the literature. In this chapter we will look at some security notions for threshold signatures first introduced by Bellare et al. in [26]. The authors of this paper, not only introduced a new hierarchy of security definitions, but they were also able to show that schemes like FROST achieve better security than what was originally believed. Another goal was to introduce a unified syntax for a specific kind of schemes, called partially non-interactive ; these are two-round schemes where the first round can be precomputed (FROST is in this scheme category as described in the previous chapter). For simplicity, we will adopt their syntax but it is easy to see how the algorithms presented in the chapter match the ones in Section 4.2. Finally, we will look at a separate paper [27] which introduced strong unforgeability for threshold signatures and how their definition of strong unforgeability compares the strong unforgeability notions of [26].

9.1 Syntax for Partially Non-Interactive Threshold Signatures

The syntax described in this chapter is strictly for partially non-interactive threshold signing schemes. These are two-round schemes where the first round can be precomputed. In these schemes we will assume that we have some servers and a leader to manage the sessions.
A partially non-interactive threshold signature scheme TS *specifies a number ns of servers, a reconstruction threshold t, a key-generation algorithm* Kg*, a server pre-processing algorithm* SPP*, a leader pre-processing algorithm* LPP*, a leader signing-request algorithm* LR*, a server partial-signature algorithm* PS*, a leader partial-signature aggregation algorithm* Agg *and a verification algorithm* Vf*. We describe how the new algorithms are used in a signing session.*

1. *Before the signing session begins, the ns servers use the* SPP *to produce presignatures/pre-processing tokens which they send to the leader.*

2. *When the leader wishes for a signing session to be started, he uses the* LR *algorithm to create a leader request lr. This leader request specifies the servers chosen to participate in the signing session as well as the pre-processing tokens they shall use.*

3. *The servers chosen use the* PS *algorithm to form partial signatures which they send to the leader.*

4. *The leader uses the* Agg *algorithm on the partial signatures to create a group signature.*

5. *Any public verifier can check the validity of said signature by using the Vf algorithm.*

9.2 Unforgeability challenges in threshold signatures

In order to define an unforgeability game for threshold signatures, the adversary needs to have access to a signing oracle. In contrast to single party signing schemes, this oracle will produce partial signatures instead of standalone signatures. This however could be problematic as, for all we know, an adversary might be able to find a way to be able to use a partial signature of a separate session to create forgeries. This naturally lead the authors of [26] to define when a threshold signature shall be considered trivial or not. For single-party signatures, a forgery is considered trivial if the oracle has been queried on that message as seen in Figure 3.1. In the game definition of Figure 4.2 for threshold signatures, we can see that a signature is considered trivial if the adversary has obtained at least one partial signature on that message from the oracle. This is natural considering that in that game, the adversary controls $t-1$ signers so all he needs is one more partial signature share to produce a group signature ; and therefore getting that share from the oracle would be trivial. However, this would not be correct if the adversary controlled less than $t-1$ signers. In order to study such cases, an unforgeability hierarchy was defined with multiple levels. For this levels it holds that : TS-UF-0 $\leftarrow$ TS-UF-1 $\leftarrow$ TS-UF-2 $\leftarrow$ TS-UF-3 $\leftarrow$ TS-UF-4 (where $B \leftarrow A$ means that any scheme which is A-secure is also B-secure). We next describe what is considered trivial in each of these levels:

- *TS-UF-0: A partial signature for the message M was generated by at least one honest server (i.e the adversary got that partial signature from the oracle).*

- *TS-UF-1: A partial signature for the message M was generated by at least $t-c$ honest servers, where c is the number of corrupted signers.*

- *TS-UF-2: There exists a leader request lr for the message M which was answered by at least $t-c$ honest signers.*

- *TS-UF-3: There exists a leader request lr for the message M such that every honest server $i \in lr.SS$ either answered lr or the pre-processing token pp_i associated with i in lr is maliciously generated.*

- *TS-UF-4: There exists a leader request lr for the message M such that every honest server $i \in lr.SS$ answered lr.*

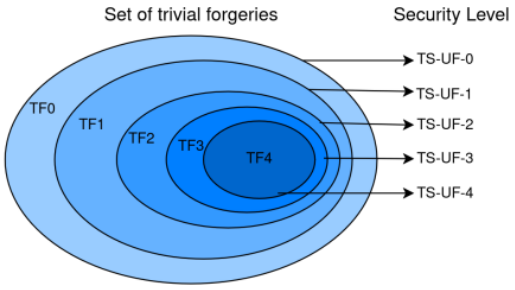

Figure 9.1: *Hierarchy of security notions*

The first two notions of unforgeability (TS-UF-0 and TS-UF-1) focus specifically on the messages, whereas the next levels focus on the leader requests. This allows more flexibility to the adversary as in levels TS-UF-2,3,4 he is allowed to mix different leader request responses (i.e different signing sessions) while trying to create a forgery. The authors also introduce the first notion of strong unforgeability for threshold signatures. In single-party schemes strong unforgeability states that a forgery (m^*, σ^*) *is non-trivial as long as* σ^* *was never output from the oracle. This means that an adversary can query the signing oracle on* m^* *as long as the final signature which he submits is not one given to him by the oracle. In order to translate this notion to the threshold setting, the authors asked that there exists an algorithm* SVF *that outputs* true *only for a single signature given a specific message. Given this algorithm, they defined three strong unforgeability levels: TS-SUF-2 ← TS-SUF-3 ← TS-SUF-4.*
In Figure 9.3 we define the unforgeability games as seen in [26]. The trivial forgery conditions appear in the following figure.

$\mathtt{tf}_0(M)$	:	$\mathrm{S}_1(M) \neq \emptyset$
$\mathtt{tf}_1(M)$	:	$\lvert\mathrm{S}_1(M)\rvert \geq t - \lvert CS\rvert$
$\mathtt{tf}_2(lr)$	:	$\lvert\mathrm{S}_2(lr)\rvert \geq t - \lvert CS\rvert$
$\mathtt{tf}_3(lr)$	:	$\mathtt{tf}_2(lr)$ and $\mathrm{S}_2(lr) = \mathrm{S}_3(lr)$
$\mathtt{tf}_4(lr)$	:	$\mathtt{tf}_2(lr)$ and $\mathrm{S}_2(lr) = \mathrm{S}_4(lr)$
$\mathtt{tsf}_2(lr, vk, sig)$	:	$\mathtt{tf}_2(lr)$ and $\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$
$\mathtt{tsf}_3(lr, vk, sig)$	:	$\mathtt{tf}_3(lr)$ and $\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$
$\mathtt{tsf}_4(lr, vk, sig)$	:	$\mathtt{tf}_4(lr)$ and $\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$

Figure 9.2: *Trivial forgery conditions and trivial strong forgery conditions.*

It is clear that the unforgeability defined in Section 4.3.2 is equivalent to TS-UF-0. The fact that FROST is TS-UF-0 secure was shown in [54]. However, in [26] it was actually shown that FROST is TS-SUF-3 secure, showing that schemes might unveil better than advertised security when looked under this new scope.

Games $\mathsf{G}^{\text{ts-uf-i}}_{\mathsf{TS}}$ $(i = 0, 1, 2, 3, 4)$ **and** $\mathsf{G}^{\text{ts-suf-i}}_{\mathsf{TS}}$ $(i = 2, 3, 4)$

INIT(CS):

Require: $CS \subset [1..n]$ and $|CS| < t$

$h \leftarrow \mathsf{TS.HF}$; $(vk, aux, sk_1, \ldots, sk_n) \xleftarrow{\$} \mathsf{Kg}[h]$

$HS \leftarrow [1..n] \setminus CS$ // Set of honest parties

For $i \in HS$ do:

$st_i.sk \leftarrow sk_i$; $st_i.vk \leftarrow vk$; $st_i.aux \leftarrow aux$

Return $vk, aux, \{sk_i\}_{i \in CS}$

PPO(i):

Require: $i \in HS$

$(pp, st_i) \xleftarrow{\$} \mathsf{SPP}[h](st_i)$; $PP_i \leftarrow PP_i \cup \{pp\}$

Return pp

PSIGNO(i, lr):

$M \leftarrow lr.\mathsf{msg}$

Require: $lr.SS \subset [1..ns]$ and $M \in \{0,1\}^*$ and $i \in HS \cap lr.SS$

Require: $lr.PP(i) \in PP_i$

$L \leftarrow L \cup \{lr\}$; $(psig, st_i) \xleftarrow{\$} \mathsf{PS}[h](lr, i, st_i)$

If $(psig \neq \perp)$ then

$S_1(M) \leftarrow S_1(M) \cup \{i\}$; $S_2(lr) \leftarrow S_2(lr) \cup \{i\}$

Return $psig$

RO(x): // Random Oracle

Return $h(x)$

FIN(M, sig):

For all $lr \in L$ do:

$S_3(lr) \leftarrow \{i \in HS \cap lr.SS : lr.PP(i) \in PP_i\}$; $S_4(lr) \leftarrow HS \cap lr.SS$

If not $\mathsf{Vf}[h](vk, M, sig)$ **then return** false

Return (not $\mathsf{tf}_i(M)$) // Game $\mathsf{G}^{\text{ts-uf-i}}_{\mathsf{TS}}$ for $i = 0, 1$

Return (not $\exists lr(lr.\mathsf{msg} = M \wedge \mathsf{tf}_i(lr))$) // Game $\mathsf{G}^{\text{ts-uf-i}}_{\mathsf{TS}}$ for $i = 2, 3, 4$

Return (not $\exists lr(lr.\mathsf{msg} = M \wedge \mathsf{tsf}_i(lr, vk, sig))$) // Game $\mathsf{G}^{\text{ts-suf-i}}_{\mathsf{TS}}$

Figure 9.3: *Games to define unforgeability for threshold signature schemes.*

9.3 Relations between notions of security

The previously defined unforgeability levels are not unrelated. In the following figure, we can see some relations between them. We refer the reader to [26] for the proofs.

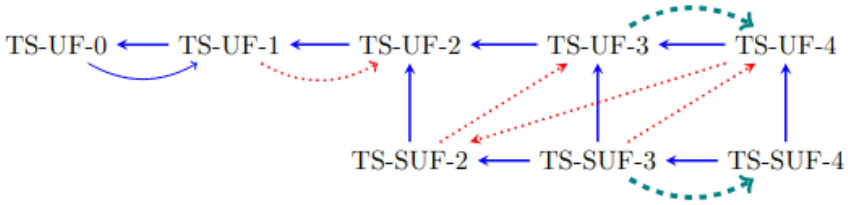

Figure 9.4: *Relations between notions of security. The blue non-dotted arrows indicates an implication under a quantitatively loose reduction. The dotted arrows show that the levels are completely separated. The thick dotted arrows indicate the existence of a generic transformation.*

9.4 Strong unforgeability for threshold signatures

In the previous section, we show that the authors of [26] defined strong unforgeability for threshold signature schemes that have a special SVf *algorithm. However, we would much prefer a definition that is closer to the one for single-party schemes and is not limited to the class of threshold signing schemes studied in [26] (i.e semi non-interactive schemes). In [27] a new definition on strong unforgeability for threshold signature schemes is given via one-more unforgeability.*

9.4.1 Strong unforgeability challenges for threshold signatures

The authors of [27] found a number of obstacles that make strong unforgeability difficult to define when it comes to threshold and multi-signatures. These issues include:

- *Pinpointing when a signature is produced: In single-party signing schemes, the signer executes a single signing operation when given a message m to sign. In contrast to this, threshold signing schemes might need multiple interactive rounds between the signing members ; these rounds are all separate steps. It is therefore difficult to pinpoint exactly when a signature gets produced. One idea would be to define that a signature has been produced once the last round of the protocol completes. Another approach would be to consider that a message m has been signed as long as a signing session on m began. This is however problematic as in reality, the signing session on m might never end. For example, in a five-round interactive protocol, it would be very problematic if the adversary can create a signature after only seeing the messages of the first round. It is clear that in that case the adversary never got to see a group signature being produced when he completed his forgery.*

- *Mixed sessions: An adversary that controls $t-1$ out of n signers might start multiple signing sessions on a message m. If this happened in the single-party setting,*

we could of course count the amount of signatures that the adversary got from the honest party and expect him to create another forgery on m in order to break strong unforgeability. In the threshold setting however, we have to consider the case where the adversary is able to mix partial signature shares from different sessions leading to multiple distinct group signatures on m even though technically none of the honest parties signed the message twice.

9.5 One-more unforgeability to the rescue

To sidestep these issues, the authors turn to a quantitative approach borrowed from blind-signature schemes: one-more unforgeability. As we have seen throughout this thesis, threshold signatures and blind signatures are very similar in the sense that the unforgeability notions of both cases allow for communication between the adversary and the honest signers and therefore it seems natural to use notions from the blind-signature literature for threshold schemes. Intuitively, the authors ask that after ℓ complete executions of the interactive signing protocol on message m, the adversary cannot output more than ℓ valid full signatures on m. In Figure 9.5 we can see the game definition of strong unforgeability for threshold signatures. From the game we can see that after an adversary executes the last interactive round of a session on m, we expect him to only be able to form a single signature on that message.

9.6 Comparing strong unforgeability with previous notions

After presenting strong unforgeability, it is natural to see how it compares to the notions of Section 9.2 and if there is any connection with them. In [27] it is shown that : TS-SUF-2 → SUF (i.e an adversary that wins the strong unforgeability of Figure 9.5, wins the TS-SUF-2 game of Figure 9.3. The proof is quite simple and we discuss it briefly:

- → *If the adversary wins the strong unforgeability game, that means he can come up with valid distinct signatures $(\sigma_j)_{j=1}^{\ell}$ for some message m s.t $\ell \geq |\{lr : lr.m = m\}|$.*
- → *Hence, there exists some signature σ_j for which $\mathsf{SVf}(pk, lr, \sigma_j) = false$ for all $lr \in \{lr : lr.m = m\}$ and thus he can use that exact signature to win in the TS-SUF-2 game.*

Games $\mathsf{G}^{\mathrm{suf-ts}}_{n,t}[\mathsf{TS}]$, $\mathsf{G}^{\mathrm{euf-ts}}_{n,t}[\mathsf{TS}]$
procedure INIT($\mathcal{CS}$) $\mathcal{CS} \subseteq \{1,\ldots,n\}$ and $|\mathcal{CS}| < t$ ▷ corrupt set
  $\mathsf{HS} \leftarrow \{1,\ldots,n\} \setminus \mathcal{CS}$ ▷ honest parties
  $\mathcal{Q} \leftarrow \emptyset$ ▷ tracks legit signatures
  $((vk_i, s_i)_{i=1}^n, \widetilde{vk}) \leftarrow \mathsf{TS.Kg}(n,t)$
  return $\widetilde{vk}$, $((vk_i)_{i=1}^n, (sk_j)_{j\in\mathcal{CS}})$
end procedure
procedure SIGNO$_j$(k, s, some subset of $\{m, S, \mathrm{out}\}$)
  if $j \notin \mathsf{HS}$ **then**
    return $\perp$
  end if
  $\sigma \leftarrow k.\mathsf{TS.Sign}(\text{input to } \mathsf{Sign}_{O_j})$
  if $\sigma = \perp$ **then**
    return $\perp$
  end if
  if $j = \mathsf{TS.ir}$ **then** ▷ on last interactive round
    if $\mathcal{Q}[s,m]$ uninitialized **then**
      $\mathcal{Q}[st_s, m] \leftarrow 1$
    else
      $\mathcal{Q}[st_s, m] \leftarrow \mathcal{Q}[st_s, m] + 1$
    end if
  end if
  return σ
end procedure
procedure FIN(m, $(\sigma_j)_{j=1}^{\ell}$)
  if $\exists\, i \neq j :\ \sigma_i = \sigma_j$ **then**
    return false
  end if
  for $j = 1,\ldots,\ell$ **do**
    if $\neg\, \mathsf{TS.Verify}(vk, m, \sigma_j)$ **then**
      return false
    end if
  end for
  if $\mathcal{Q}[m]$ initialized $\wedge\ \mathcal{Q}[m] \geq \ell$ **then**
    return false
  end if
  if $\mathcal{Q}[m]$ initialized **then**
    return false
  end if
  return true
end procedure

Figure 9.5: *Strong unforgeability for threshold signatures. The lines in blue are for strong unforgeability while the red lines are for existential unforgeability.*

Chapter 10

Threshold signature schemes against Adaptive Adversaries

10.1 Static vs Adaptive Security

Throughout this entire thesis we've only looked at security against static adversaries. That means that the adversary has to declare which parties he wishes to corrupt at the beginning of the unforgeability game. However, this poses a clear limitation on his abilities. In a real-world example, the adversary would be able to monitor the signing parties and look at their messages, then he would able to hack the parties he wishes only after having seen their messages. While sticking to static security is a much simpler goal, as threshold signature schemes aim to be approved for real-world application, finding schemes that are unforgeable against an adaptive adversary is of great importance. This is evident from the threshold signature schemes call that NIST made [11]. It is cleared stated there that "a proposed protocol must not allow its critical safety properties to be trivially broken in case of adaptive corruptions". In this chapter we will look at two schemes, Sparkle and Glacius which provide security under adaptive corruptions and compare them ; but first, let us define the adaptive unforgeability game as seen in [29]. Note than in the literature for adaptive threshold signatures, it is common to symbolize the threshold as $t + 1$ instead of t, so we will stick to that. From Figure 10.1 we can see that in this version of the game, the adversary has access to a corruption oracle allowing him to corrupt singers whenever he wishes as long as he does not corrupt more than t members. Whenever a party gets corrupted, the adversary learns their secret key and their internal state. Our desired goal would be to find schemes unforgeable even if the adversary can corrupt t signers in an adaptive manner.

10.2 The Sparkle+ scheme

The first threshold Schnorr scheme to be proven secure against adaptive corruptions was the Sparkle+ scheme [28] (which we call Sparkle for simplicity). This scheme is a simple three-round scheme that very much resembles the construction of MuSig1 presented in Section 6.2. The Sparkle scheme does however achieve full adaptive security in the ROM where the authors were able to prove security when the adversary has $t/2$ available

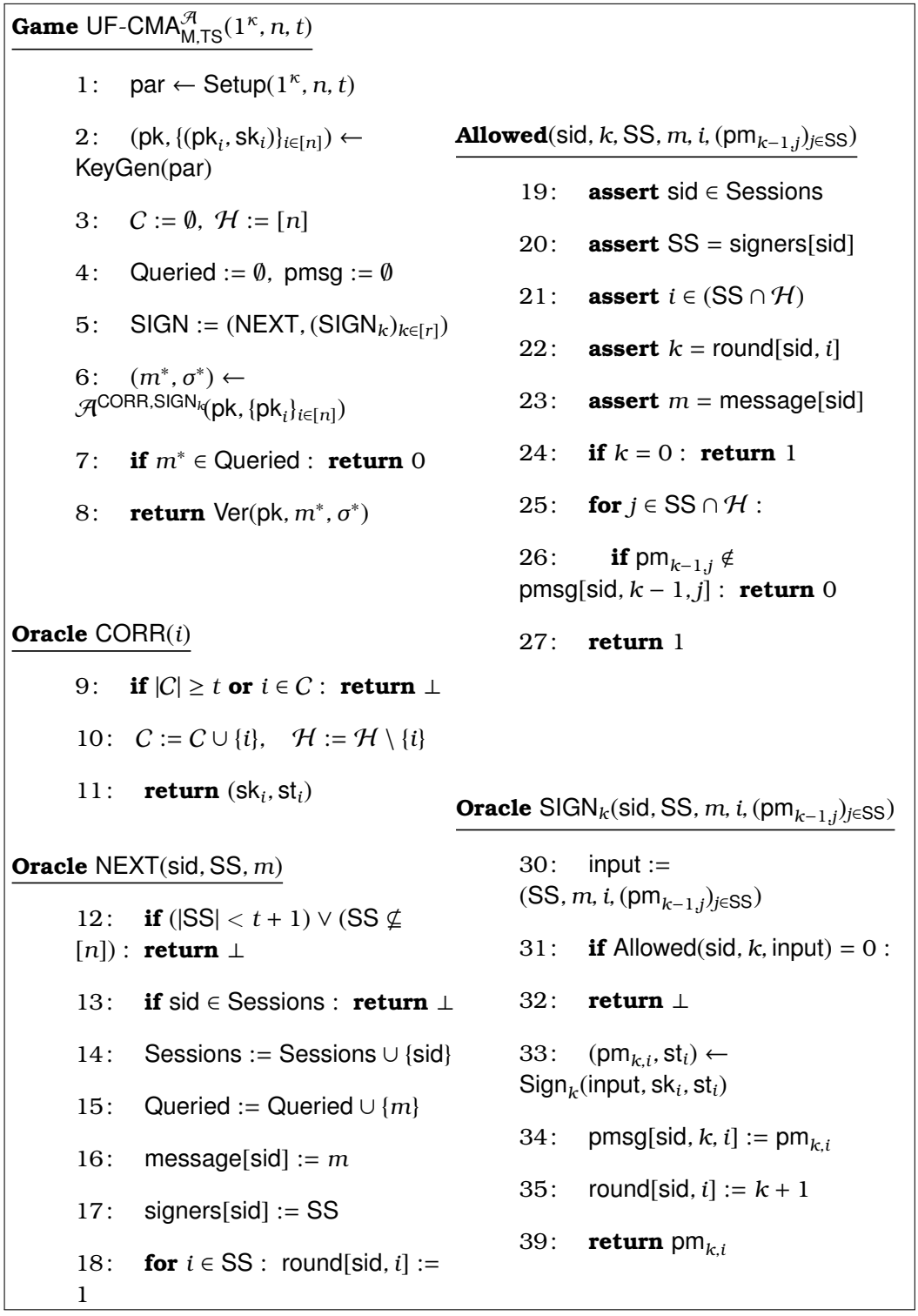

Figure 10.1: *UF-CMA game for an r-round threshold signature scheme against adaptive adversary*

corruptions. However, when moving to the AGM, Sparkle achieves full adaptive security. This is because, without the AGM, the authors had to resort to the forking lemma [30] in order to prove security. This is problematic when it comes to adaptive adversaries as if we allowed the adversary to corrupt t members, that means he would be able to corrupt $t + t = 2t$ parties throughout the proof (the fact that the adversary is adaptive means that the t corrupted parties of the first run and the t corrupted parties of the second run might be different) leading to trivial forgeries. For this reason, when using the forking lemma, the authors allow the adversary to corrupt $t/2$ participants, leading to a maximum number of t corrupted members in total.
Please note that the original version of Sparkle [55] was not secure under a specific attack presented in [56]. The authors of Sparkle fixed the issue by including a separate single-party signature for each protocol message exchanged during Sparkle. This however, is no issue for us as we always assume that protocol messages are sent with a signature as described in Section 4.1.4.

10.2.1 Algorithms of the Sparkle Scheme

- *Setup. We work in a group $\mathbb{G}$ of order p and g is a generator of the group. The scheme will also need two hash functions $H_{sig}, H_{com} \rightarrow \mathbb{Z}_p$.*

- *KeyGen. We need to use a DKG algorithm like PedPoP with n signers and threshold t. At the end of the protocol, each signer holds a secret key x_i, a public key X_i and the group has an aggregate key $\tilde{X}$.*

- *Sign$_1$: We assume the signers wish to sign an (agreed-on) message m. We also assume that the signing set SS of the session is known to the participants. Each signer P_i generates a random $r_i \xleftarrow{\$} \mathbb{Z}_p$, computes $R_i = g^{r_i}$, $t_i = H_{com}(R_i)$ and sends out t_i to all other signers in SS.*

- *Sign$_2$: After receiving all $\{t_i\}_{i \in SS}$ from all other cosigners, participant P_i reveals his commitment nonce R_i by sending it out to all signers.*

- *Sign$_3$: After receiving all $\{R_i\}_{i \in [n]}$ from all other cosigners, participant P_i checks that $t_i = H_{com}(R_i)$ and aborts the protocol if this is not the case. Otherwise, he computes the group commitment $R = \prod_{i \in SS} R_i$, the challenge of the Schnorr signature $c = H_{sig}(\tilde{X}, R, m)$ and his partial signature share $z_i = r_i + c \cdot \lambda_i \cdot x_i$ which he sends out to all other signers (or the central coordinator role).*

- *Combine. After all partial signatures $\{z_i\}$ have been sent out and verified, the final group signature on m is produced as $\sigma = (R, \sum_{i \in SS} z_i)$.*

- *Verify. Any public validator can verify the signature under the public key $\tilde{X}$ by using the Schnorr single party verification algorithm on σ.*

Setup(1^κ)

$(G, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

$H_{com}, H_{sig} : \{0,1\}^* \rightarrow \mathbb{Z}_p$

$par \leftarrow ((\mathbb{G}, p, g), H_{agg}, H_{com}, H_{sig})$

return par

KeyGen(t, n, par)

$(\tilde{X}, \{(X_j, x_j\}_{j\in[n]}) \leftarrow \mathrm{PedPoP}(t, n)$

for j = 1 **to** n:

$pk_j \leftarrow X_j$

$sk_j \leftarrow x_j$

return $\tilde{X}, \{pk_j\}_{j\in[n]}, \{sk_j\}_{j\in[n]}$

Sign$_1(SS, m, i, \{pm_{0,j}\}_{j\in SS}, sk_i, st_i)$

$r_i \leftarrow \mathbb{Z}_p$

$R_i \leftarrow g^{r_i}$

$t_i \leftarrow H_{com}(R_i)$

$pm_{1,i} \leftarrow t_i$

$st_i \leftarrow (r_i, R_i)$

return $pm_{1,i}, st_i$

Sign$_2(SS, m, i, \{pm_{1,j}\}_{j\in SS}, sk_i, st_i)$,

parse $(R_i, r_i,) \leftarrow st_i$

parse $\{t_j\}_{j\in SS} \leftarrow \{pm_{1,j}\}_{j\in SS}$

$pm_{2,i} \leftarrow R_i$

$st_i \leftarrow (R_i, r_i, \{t_j\}_{j\in SS})$

return $pm_{2,i}, st_i$

Sign$_3(SS, m, i, \{pm_{2,j}\}_{j\in SS}, sk_i, st_i)$,

parse $(R_i, r_i, \{t_j\}_{j\in SS}) \leftarrow st_i, x_i \leftarrow sk_i$

parse $\{R_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

for j $\in SS$

if $H_{com}(R_j) \neq t_j$

return $\perp$

$R \leftarrow \prod_{j\in SS} R_j$

$c \leftarrow H_1(R, \tilde{X}, m)$

$z_i \leftarrow r_i + c \cdot \lambda_i \cdot x_i$

$pm_{3,i} \leftarrow z_i$

return $pm_{3,i}, st_i$

Combine$(\{pm_{k,j}\}_{k\in[3],j\in SS})$

parse $\{R_j\}_{j\in SS} \leftarrow \{pm_{2,j}\}_{j\in SS}$

parse $\{z_j\}_{j\in SS} \leftarrow \{pm_{3,j}\}_{j\in SS}$

$R \leftarrow \prod_{j\in SS} R_j$

$c \leftarrow H_{sig}(R, \tilde{X}, m)$

for j $\in SS$

if $g^{z_j} \neq R_j \cdot X_j^{\lambda_j \cdot c}$

return $\perp$

$z \leftarrow \sum_{i\in SS} z_i$

$R \leftarrow \prod_{i\in SS} R_i$

$\sigma \leftarrow (R, z)$

return σ

Verify(σ, Y, m)

parse $\sigma \leftarrow (R, z)$

$c \leftarrow H_1(R, Y, m)$

return 1 **if** $g^z = R \cdot Y^c$, **else** 0

Figure 10.2: *The Sparkle+ scheme*

10.2.2 Complexity analysis of Sparkle

For each session, each signer has to send out elements of size:

- ***Sign***$_1$*:* $n \cdot \mathbb{Z}_p$ *(broadcast)*
- ***Sign***$_2$*:* $n \cdot \mathbb{G}$ *(broadcast)*
- ***Sign***$_3$*:* $\mathbb{Z}_p$ *(sent to coordinator)*

Thus, the total amount of communication for a Sparkle sessions is: $n \cdot ((n+1) \cdot \mathbb{Z}_p + n \cdot \mathbb{G})$. *For each session, each signer has to perform computations of cost:*

- ***Sign***$_1$*:* $GExp + Hh$
- ***Sign***$_2$*:* -
- ***Sign***$_3$*:* $(t+1) \cdot Hh + (t-1) \cdot GMul + 2 \cdot SMul + Lagr$

Therefore, the total amount of computation for each signer is $(t+2) \cdot Hh + GExp + (t-1) \cdot GMul + 2 \cdot SMul + Lagr$.

10.3 The Glacius scheme

As we saw, the Sparkle protocol cannot achieve full adaptive security in the ROM. Glacius [29] on the other hand, is a five-round threshold Schnorr scheme which achieves full adaptive security in the ROM under the DDH assumption.

10.3.1 Algorithms of the Glacius schemes

- *Setup. We work in a group* $\mathbb{G}$ *of order* p *and* g *is a generator of the group. The scheme will also need five hash functions* $H_0, H_1, H_{sig}, H_{com}, H_{view}$.

- *KeyGen. They key generation algorithm takes as input the public parameters and samples three uniformly random polynomial* $s(x), r(x), u(x) \xleftarrow{\$} \mathbb{Z}_p[x]_{(t)}$ *such that* $r(0) = u(0) = 0$ *where* $\mathbb{Z}_p[x]_{(t)}$ *is the set of all polynomial in* $\mathbb{Z}_p$ *of degree* t. *The secret signing key of signer* i *is then* $sk_i \leftarrow (s(i), r(i), u(i))$ *and its public key share is* $pk_i = g^{s(i)} h^{r(i)} v^{u(i)}$. *Further, the public key of the system is* $pk = g^{s(0)} h^{r(0)} v^{u(0)} = g^{s(0)}$.

- $Sign_1$. *We assume the signers wish to sign an (agreed-on) message* m. *We also assume that the signing set SS of the session is known to the participants. Each* P_i *samples a uniformly random string* $\rho_i \xleftarrow{\$} \{0,1\}^\kappa$ *which he broadcasts.*

- $Sign_2$. *Upon receiving all random strings* $\vec{\rho} = ((j, \rho_j))_{j \in SS}$ *from signers, each singer* P_i *proceeds as follows. He samples a random* $a_i \xleftarrow{\$} \mathbb{Z}_p$ *and computes the nonce* $A_i \leftarrow (g^{a_i} \cdot H_0(\vec{\rho})^{r(i)} \cdot H_1(\vec{\rho})^{u(i)})^{\lambda_i}$ *where* λ_i *is the Lagrange coefficient. Signer* P_i *then broadcasts his commitment* $\mu_i = H_{com}(i, A_i)$.

- *$Sign_3$. Upon receiving all commitments $\vec{\mu} = (\mu_j)_{j \in SS}$ from signers, each P_i hashes its current view of the protocol messages $(\vec{\rho}, \vec{\mu})$ and computes the hash $y_i = H_{view}(\vec{\rho}, \vec{\mu})$ which he broadcasts.*

- *$Sign_4$. Upon receiving all views $\vec{y} = (y_j)_{j \in SS}$ from signers, each P_i proceeds as follows. For all $j \in SS$, he checks whether $y_j = y_i$ holds i.e checks whether his view matches with the views of all other honest signers. If one of these checks fails, the signer outputs $\perp$ and aborts. Otherwise, he sends the opening A_i to all other signers.*

- *$Sign_5$. Upon receiving all opening $(A_j)_{j \in SS}$ from signers, each P_i proceeds as follows. First, it retrieves the commitments $\vec{\mu} = (\mu_j)_{j \in SS}$ from the second round. Then, for all $j \in SS$, it checks whether $\mu_j = H_{com}(j, A_j)$ holds. If either of these checks fails, he outputs $\perp$ and aborts. Otherwise, he computes the combined nonce $\hat{A}_i = \prod_{j \in SS} A_j$, challenge $c = H_{sig}(\hat{A}, pk, m)$ and his signature $z_i = \lambda_i(a_i + c \cdot s(i))$.*

- *Combine. After all partial signatures $\{z_i\}$ have been sent out and verified, the final group signature on m is produced as $\sigma = (\prod_{i \in SS} A_i, \sum_{i \in SS} z_i)$.*

- *Verify. Any public validator can verify the signature under the public key $\tilde{X}$ by using the Schnorr single party verification algorithm on σ.*

10.3.2 Complexity analysis of Glacius

For each session, each signer has to send out elements of size:

- ***Sign***$_1$*: $n \cdot \kappa$ (broadcast)*
- ***Sign***$_2$*: $n \cdot \mathbb{G}$ (broadcast)*
- ***Sign***$_3$*: $n \cdot \kappa$ (broadcast)*
- ***Sign***$_4$*: $n \cdot \mathbb{G}$ (broadcast)*
- ***Sign***$_5$*: $\mathbb{Z}_p$ (sent to the coordinator)*

Thus, the total amount of communication for a Glacius session is: $n \cdot (2 \cdot n \cdot \mathbb{G} + 2 \cdot n \cdot \kappa + \mathbb{Z}_p)$. For each session, each signer has to perform computations of cost:

- ***Sign***$_1$*: –*
- ***Sign***$_2$*: $2 \cdot Hh + Lang + 3 \cdot GExp + 2 \cdot GMul$*
- ***Sign***$_3$*: Hh*
- ***Sign***$_4$*: –*
- ***Sign***$_5$*: $2 \cdot Hh + 2 \cdot SMul + (t-1) \cdot GMul$.*

Therefore, the total amount of computation for each signer is: $5 \cdot Hh + 3 \cdot GExp + (t+1) \cdot GMul + 2 \cdot SMul + Lagr$.

10.4 Sparkle vs Glacius

Scheme	Rounds	Communication (total)	Computation (per signer)
Sparkle	3	$n \cdot ((n+1) \cdot \mathbb{Z}_p + n \cdot \mathbb{G})$	$(t+2) \cdot Hh + GExp + (t-1) \cdot GMul + 2 \cdot SMul + Lagr$
Glacius	5	$n \cdot (2 \cdot n \cdot \mathbb{G} + 2 \cdot n \cdot \kappa + \mathbb{Z}_p)$	$5 \cdot Hh + 3 \cdot GExp + (t+1) \cdot GMul + 2 \cdot SMul + Lagr$

Figure 10.3: *Comparison of Sparkle and Glacius*

10.5 New advances in threshold signatures against adaptive corruptions

Concurrently with this thesis, two very interesting papers on the adaptive security of threshold signatures were published. In [57] it is proven that it is impossible to prove the adaptive security of any key-unique threshold signature scheme under any non-interactive computational assumption above $t/2$ corruptions, for a broad class of reductions. It is also proven that it impossible to prove the adaptive security of any key-unique threshold Schnorr signature under (A)OMDL in the ROM above $t/2$ corruptions via rewinding. This makes sense of Sparkle's $t/2$ adaptive corruption threshold. In [58] the FROST schemes are proven secure against adaptive adversaries in the ROM with a corruption threshold $t/2$ and when moving to the AGM, FROST is secure under a full adaptive threshold under AOMDL and a new assumption called low-dimensional vector representation problem.

Chapter 11

Sparkling ROAST: A robust wrapper protocol for Sparkle+

As we've already discussed, robustness is vital for real-world systems where denial of service attacks are costly. Recalling its definition, robustness simply states that a protocol is guaranteed to output a valid signature in the presence of t honest signers, even if the remaing signers try to prevent the protocol from completing. As real-world applications aim to adopt threshold signatures, robustness in Schnorr schemes should be a main goal if we wish Schnorr schemes to be considered. The well established security of Schnorr signatures is not going to be enough when other candidate schemes have robustness as an extra property. In the case of FROST [21] [24], we saw how the ROAST [25] wrapper protocol can be used in order to add robustness to the initial threshold signing protocol. However, the construction of ROAST only works for a specific kind of two-round schemes with identifiable abort [25] [29]. It is natural to consider whether this wrapper protocol can be extended to other protocols which include more rounds, the simplest one being Sparkle+ [28] (which we call Sparkle for simplicity). In this chapter, we discuss an extension of ROAST for Sparkle, which we call Sparkling ROAST, justifying why it achieves robustness and how many internal Sparkle sessions will have to be run for success in finite time. Just like in ROAST, we work in the asynchronous model where the only guarantee that we have is that protocol messages between signers will reach their destination eventually, but without being able to set a specific time frame (robustness is trivial in the synchronous model by raising timers against delaying participants). In order to do so, we study the original ROAST and draw parallelisms in order to extend it in this new setting. Finally, we argue that our new construction can be extended for even more schemes as long as said schemes have the Identifiable Abort property (an example would be the five-round Glacius scheme [29]).

11.1 ROAST on FROST

In this section, we look at the ROAST protocol while trying to identify some key information on why it provides robustness for FROST and on how many internal sessions need to run until termination.

ROAST works by having a coordinator orchestrate multiple internal FROST sessions until

one of them succeeds (deterministically). In order to do so, the coordinator maintains a list of responsive signers i.e signers that have responded to all previous signing requests. As soon as that list contains t signers, the coordinator will start a signing session of FROST with them i.e ask each signer to respond with a valid signature share. Along with every partial signature, every signer is required to send a fresh presignature share in preparation of a possible next signing session (we refer to this technique as piggybacking). This way a pipeline of signing sessions is created as signers who answer honestly will always have an available presignature to join a new session if needed. For FROST, presignatures are of the form $(D = g^d, E = g^e)$. We consider that each signing session begins with the coordinator sending a list of presignatures and the signing parties included in the signing set SS i.e if a participant P_i receives the list $\{(j, D_j, E_j)\}_{j \in SS}$ he is supposed to provide a partial signature based on that signing committee and those presignatures. Of course it is assumed that in the previous list, the values (D_i, E_i) are a presignature which the signer sent earlier to the coordinator otherwise it would be impossible for him to find the discrete logarithms needed for the partial signature. Finally, we assume that the coordinator and the signers know which message is indented for which session.

ALGORITHM 11.1: *ROAST Protocol for FROST*

Signer Behavior:
Upon init:
    Send initial presignature share.
Upon receiving a presignature share (start of new session):
    Send partial signature.
    Send new presignature share.

Coordinator Behavior:
Upon init:
    Mark all nodes as UNRESPONSIVE.
Upon receiving an initial presignature share:
    Mark sender as RESPONSIVE.
Upon receiving a partial signature and new presignature share for session *sid*:
if partial signature does not verify **then**
        Mark sender as MALICIOUS and exclude them from future sessions.
end if
if session *sid* has t signature shares **then**
        Compute and output group signature.
end if
Mark sender as RESPONSIVE.
if there are t RESPONSIVE members **then**
        Start a new session with them.
        Send them their presignatures.
        Mark them as UNRESPONSIVE.
end if

Overall, the new FROST protocol for signing parties looks as follows:

- *When ROAST begins: Send initial presignature.*

- *Added to signing session: Send partial signature based on presignatures,* ***and new presignature.***

The fact that FROST is a partially non-interactive protocol (meaning that it includes two-rounds the first of which is message independent) means that the ROAST construction allows us to bring it down to a single interactive round by including a presignature with each partial signature share (pretty much reducing the protocol rounds to one). This introduces a single blocking point in the pipeline (i.e a single point to determine which participants are honest): when a party enters a signing session, he will either answer or not ; of course due to the asynchronous setting we do not know when (or if) that will happen. However since honest participants are guaranteed to answer, we can separate participants into two distinct sets: Responsive and Unresponsive:

- *Responsive : These participants are not blocking any sessions from concluding. Also, they have an available presignature making them available to join a new session whenever needed.*
- *Unresponsive : These participants are in a signing session but have not provided their partial signature share. We do not know whether these members are adversarial parties looking to perform a denial of service attack or honest members whose partial signature transmission is delayed.*

To get a better visual understanding of the two states we present the following state machine (adding a new Starting state for simplicity).

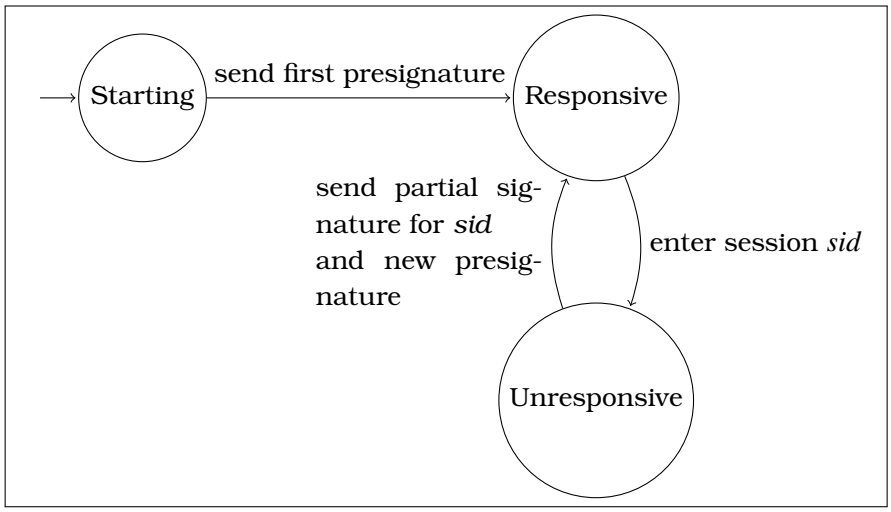

Figure 11.1: *State machine for ROAST*

To justify why the ROAST protocol provided robustness for FROST we look at some key properties:

1. ***Honest signers will never stay at Unresponsive state indefinitely**. By definition, honest signers always answer with valid signatures and their protocol messages will eventually reach the coordinator. That means that whenever they are added to a signing session (and are thus put on Unresponsive state) they will always reach*

Responsive state eventually, which also means that they will be available to be added to further sessions.

2. ***An adversarial member cannot escape a signing session undetected***. *Due to the Identifiable Abort property of FROST, we can tell when an adversary sends out a non-valid partial signature. Of course, parties that send invalid responses are excluded from future sessions.*

3. ***Adversaries are Unresponsive forever***. *From the previous observation we can conclude that an adversary added to a signing session will be added to the Unresponsive set and never be able to escape. The reason for that is in order to escape that state, the adversary will need to reply with a partial signature. If that partial signature is non-valid he is kicked out whereas if it passes validation, the adversary acted as an honest signer and we will gladly accept his partial signature to form a group signature. Since his goal is to stop a group signature from being formed, his best bet is therefore to stay at Unresponsive state forever, completely blocking that session from terminating.*

4. ***Invariant: Each party can block at most one session***. *Due to the construction of the wrapper protocol, an Unresponsive party will never join a new session. That means that the most that a single party can block is a single session.*

From the previous observations we can prove that ROAST terminates via a contradiction argument. Let us assume that no internal signing session terminates. By observation 1, we know that honest signers will eventually (and constantly) be added to the Responsive set. Since, there are at least t honest signers present (we could consider exactly t honest signers present), by construction we will eventually always have t parties in the Responsive set => a new session can always be started. As a result there will eventually be $f + 1$ sessions during the execution. Let's consider now the point at which the $f + 1$ session is initiated (where f is the number of adversarial signers). By the invariant (observation 4) as well as observations 2 and 3, we know that at most f malicious members are Unresponsive in at most one session each. Thus, among the $f + 1$ sessions there exists a session in which all signers are honest. This session is guaranteed to succeed and we've therefore reached a contradiction. Thus, the ROAST protocol will always terminate in a signature.

Finally, we've also proven that ROAST needs at most $f + 1$ internal FROST sessions. Since we have t honest signers present, it holds that $f \leq n - t$ and of course it holds that $f \leq t - 1$, otherwise the adversarial parties can trivially produce signatures on behalf of the group. Therefore, the maximum amount of internal sessions is $n - t + 1$ or t depending on which assumption we prefer.

11.2 Extending ROAST for Sparkle+

Since Sparkle [28] is also a scheme with Identifiable Abort, we believe that a new wrapper protocol, an extension of ROAST should be able to add robustness to the original

protocol. The main difference between Sparkle and FROST (and the reason why the original ROAST does not work on Sparkle) is of course the fact that the former is a three-round protocol whereas the latter is two-rounds. The fact that FROST's first round is message independent allowed the wrapper protocol to reduce the rounds to one by piggybacking a presignature along with every signature share. In Sparkle, we have that the first round (in reality the second one as well) is message independent. By using the same piggybacking approach (our new presignatures will be hashes of the form H(D)) therefore we see a clear way to reduce the protocol rounds to two, which is not enough for us to apply ROAST on it. We either have to reduce the rounds further or extend the ROAST protocol in some way. Reducing the rounds of Sparkle however is completely out of the question as that would be equivalent to completely removing the initial round (where each signer member provides a hash value). This would of course be insecure in concurrent sessions [15] just like the original version of MuSig [16]. Our approach should therefore be to use the piggybacking approach to reduce the amount of interactive rounds to two, and then introduce a new version of ROAST that will work for our underlying scheme.

11.2.1 Challenges of multi-round robustness

We recall that in FROST, the fact that piggybacking allowed us to have only one interactive round introduced a single blocking point. In Sparkle however, since the amount of interactive rounds will be reduced to two, introduces two blocking points :

1. *Blocking point on the second round of Sparkle: A party could block a session by not revealing the preimage D to their presignature $\rho = H(D)$.*

2. *Blocking point on the third round of Sparkle: A party could block a session by not sending out their partial signature during the third round.*

As we will see, the fact that there are two blocking points instead of one makes it impossible to stop parties from joining multiple sessions at the same time. Intuitively, this means that while in ROAST each adversary can block at most one internal session, this will not be the case here. An easy reason to see why this cannot work is the following scenario: Let's consider that we only allow signers to join a new session once they send out their partial signature for their current session. This allows the adversary to completely break the availability of the protocol by just not revealing their hash preimage during the second round of Sparkle. In this case, honest signers cannot compute partial signatures as all preimages need to be revealed for them to calculate their signature share.

One might say that since in the FROST version of ROAST , where only a single blocking point is present, each party can block at most one session ; then in our extended version of ROAST for Sparkle, where we now have two blocking points, each adversary should be able to block at most two sessions, one for each blocking point. This would be a desirable goal but is not so simple as we will see.

Even after agreeing that each party should be able to join multiple sessions at the same time, it is not exactly clear when the piggybacked presignatures should be sent. One (flawed) approach would be to have each signer send out their fresh presignature along with their partial signature share, just like in the original ROAST. This would however easily be attacked by an adversary who once again refuses to reveal the preimage to their presignature during the second round of Sparkle. To combat this, we argue that the fresh presignatures should be sent when revealing the preimage instead of when creating the partial signature share. This way honest signers should always have one extra available presignature per session they get added in.

11.2.2 What we achieve

We construct the Sparkling ROAST protocol, a wrapper protocol on Sparkle which adds robustness. We present two versions of the protocol, one which guarantees robustness with t honest participants and needs at most $f \cdot (n - t + 2) + 1$ internal sessions, and one which guarantees robustness in at most $2 \cdot f + 1$ sessions but requires there are $\frac{f \cdot (t-1)}{2} + t$ honest participants present (instead of t). Finally, we briefly describe how this protocol can be extended to other threshold signature schemes so long as those schemes have the Identifiable Abort property.

11.2.3 Sparkling ROAST construction

We once again assume that we are in the asynchronous setting and that the coordinator and the members are able to distinguish which message is sent for which session (this can be done by including the session ID sid with every sent message). The new presignatures of Sparkle will be of the form $H(D)$ where $D = g^d$ is the preimage of the presignature. We consider that each signing session begins with the coordinator sending a list of presignatures and the signing parties included in the signing set SS i.e if a participant P_i receives the list $\{j, H(D_j)\}_{j \in SS}$ he is supposed to follow the signing algorithm with that signing committee and which those presignatures in mind. Of course, it is assumed that in the previous list, the presignature $H(D_i)$ is a presignature which the signing party sent to the coordinator at an earlier stage otherwise it would be impossible for him to find the preimage to the hash.

As we mentioned earlier, two blocking points means two Unresponsive states : one that demonstrates that a signer is blocking the second round from concluding (i.e doesn't send the preimage corresponding to the hash value) and one that demonstrates that a signer is blocking the third round from concluding (i.e doesn't send out their partial signature). We call these new states Blocking2 and Blocking3 in order to demonstrate which round of the session they are holding (second and third round of Sparkle respectively).
The Responsive state however is also problematic. A signer should not only be Responsive when they send out their partial signature but also when they reveal their preimage during the second round (otherwise an adversarial signer that stays on state Blocking2 would hold the honest members hostages and block them from joining future sessions). Therefore, if someone reveals their preimage during the second round of the protocol and is waiting for

the other parties to do as well, they should be put on a new semi-Responsive state which we call Pending. We should start new sessions for signers that are Responsive or Pending for some session but are not Blocking2 or Blocking3 in any sessions. Of course, in order to add a member to a new session, that member should have an available presignature so the coordinator will also have to keep track of members with available presignatures ready.

Overall, the new Sparkle protocol for signing parties looks as follows:

- *When Sparkling ROAST begins: Send initial presignature.*
- *When added to a signing session: Reveal preimage to presignature* ***and send new presignature****.*
- *When all preimages have been revealed: Send partial signature based on the preimages.*

The coordinator's job will be to:

- *Maintain the state of each participant for each session, starting sessions whenever t signers exist in his set PotentialSigners (we explain what PotentialSigners is a bit).*
- *Detect and kick out malicious signers (via Identifiable Abort). This means:*
  - *Make sure that preimages match presignatures i.e for a signing session where participant P_i joined with presignature ρ_i and revealed preimage D_i, make sure that $H(D_i) = \rho_i$.*
  - *Make sure that partial signatures for specific sessions verify.*

We describe the algorithm for Sparkling ROAST in Figure 11.2.
Each state demonstrates something different:

- *Responsive : This is a session independent state. A server on this state has either finished a session or never been in one (has only sent the starting presignature).*
- *Blocking2 : A session specific state. As soon as a server enters a new session he transitions to this state. At this state we are waiting for the server to reveal his preimage corresponding to the presignature used for this session.*
- *Pending : A session specific state. A server on this state has revealed his preimage and is waiting for the other signers to do so as well.*
- *Blocking3 : A session specific state. As soon as all the preimages are revealed (i.e all signers went from Blocking2 $\rightarrow$ Pending) , all signers get moved to this state until they provide their partial signature for this session.*

The state machine for this new protocol can be seen below:

ALGORITHM 11.2: *Sparkling ROAST protocol*

Signer Behavior:

Upon init:
    Send initial presignature share.
Upon receiving a presignature share (start of new session, round 2 of Sparkle):
    Send preimage of presignature.
    Send new presignature share.
Upon receiving all preimages(round 3 of Sparkle):
    Send partial signature.

Coordinator Behavior:

Upon init:
    Mark all nodes as STARTING.
Upon receiving an initial presignature share from P_i**:**
    Mark P_i as RESPONSIVE.
Upon receiving a preimage of presignature and new presignature share for session *sid* **from** P_i**:**
if preimage is not the preimage of presignature **then**
        Mark P_i as MALICIOUS and exclude them from future sessions.
else
        Remove P_i from BLOCKING2[sid].
        Mark P_i as PENDING[sid].
        Perform CheckNewSession procedure.

    if |Pending[*sid*]| = t **then**
            Send Preimages[*sid*] to the signers of sessions *sid*.
            Remove these signers from PENDING[*sid*].
            Mark the signers of *sid* as BLOCKING3[*sid*].
    end if
end if
Upon receiving a partial signature and new presignature share for session *sid* **from** P_i**:**
if partial signature does not verify **then**
        Mark P_i as MALICIOUS and exclude them from future sessions.
end if
if session *sid* has t signature shares **then**
        Compute and output group signature.
end if
Remove P_i from BLOCKING3[*sid*].
Mark P_i as RESPONSIVE.

Procedure: CheckNewSession

procedure CHECKNEWSESSION
    if there exists at least t parties in *PotentialSigners* **then**
        Start a new session with those signers.
        Mark those signers as BLOCKING2[*sid*].
    end if
end procedure

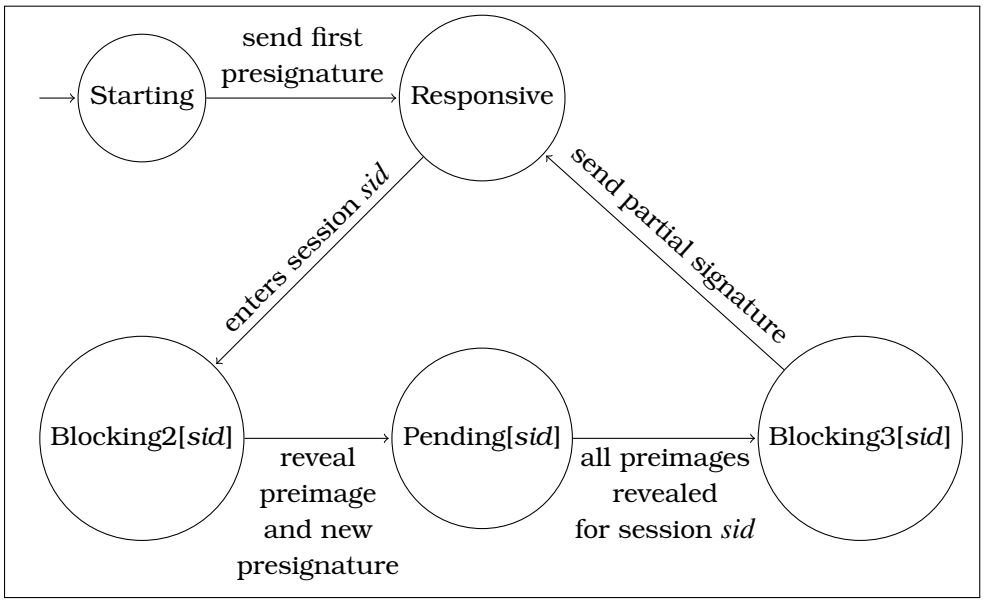

Figure 11.2: *State machine for Sparkling ROAST*

The only thing left to do is to define the set PotentialSigners which the coordinator has to check in order to see if a new session can be started. We define:

$$PotentialSigners = \left\{ P_i \;\middle|\; \begin{array}{l} \neg\exists sid : P_i \in Blocking2[sid] \\ \wedge\ \neg\exists sid : P_i \in Blocking3[sid] \\ \wedge\ Available\_PreSigs[P_i] \geq 1 \end{array} \right\} \tag{11.1}$$

This means that we allow signers to join new session so long as they are not in a Blocking state. Based on this PotentialSigners set and the pseudocode we can write down some observations (just like we did for the original ROAST) to allow us to see whether this new protocol provides robustness and if so, how many internal sessions need to be run:

1. ***Honest signers will never stay at Blocking2 or Blocking3 states indefinitely**. By definition, honest signers always answer with valid preimages and valid signatures and their protocol messages will eventually reach the coordinator. That means that whenever they are added to a signing session (and are thus put on Blocking2 state for that session) they will always reach Pending state eventually. The same will happen when they reach Blocking3 state, they will provide a valid partial signature share escaping the Blocking state.*

2. ***An adversarial member cannot escape a signing session undetected**. Due to the Identifiable Abort property of Sparkle, if a preimage that does not match the presignature is revealed or a partial signature that does not verify is sent then that signer is automatically considered malicious and kicked out of future sessions.*

3. ***If an adversary reaches Blocking3 for some session he is stuck there indefinitely**. This stems from the previous observation. In Blocking3 state we are awaiting for the adversary to send out his partial signature. The adversary cannot send a non-verifying partial signature as he will be detected as malicious. Therefore, his*

best bet is to stay at Blocking3 state forever, completely blocking that session from terminating.

4. ***Adversaries whose strategy is to block sessions at Blocking2 state will never join future sessions***. *Due to the construction of the PotentialSigners set, if someone is at Blocking2 state he will never be able to join a separate session.*

From observation 1 we can infer that (since t honest parties are present), these honest signers will always eventually (and constantly) be added to PotentialSigners and then into new signing sessions. This means that new sessions can always be created. The question is what is the maximum amount of sessions that an adversary can block (and is that maximum finite in order to prove robustness?). From observation 4 it is clear that an adversary who will not reply at all will only block a single session. Therefore, the adversary's strategy cannot be so simple. On the other hand, if he replies and joins Pending state, he will be able to join a new session until he reaches Blocking state either in this session or a future one (for example when an adversary gets added to a session where the rest of the signing parties are honest, the amount of time that he is in Pending state will depend on how long it will take the honest signers to also reach that state. As soon as that happens, the adversary will be Blocking3). Therefore it seems that the amount of sessions an adversary can block depends on the latency between the other signers and the coordinator. In order to count the maximum amount of sessions (and create a new invariant for our protocol) we consider the following worst-case scenario:

We assume maximum latency between the honest signers and the coordinator and minimum latency between the adversaries and the coordinator. The reason we need that, is because we will have the adversary reveal his preimage during the second round (as staying at Blocking2 state will exclude him from future sessions), the issue with that is that when all preimages have been revealed for a session in which he is in, he will automatically be moved to Blocking3 state for that session (and be blocked from future sessions according to observation 3). The best bet of the adversary is to escape Blocking2 state as soon as possible and join Pending state in order to be re-added to PotentialSigners. We assume than whenever a session gets created and the adversary is available, he will join this new session. We will look at how many sessions an adversary is able to block in this setting to find our invariant.

Assume Adversaries $= \{1\}$ *and Honest* $= \{2, 3, \cdots, n\}$*:*

1. ***Session 1***: *Signers* $= \{1, 2, \cdots, t\}$ *; The adversary quickly answers to escape Blocking2 state. At some point only a single Blocking2 signer is left on this session, let's assume that is signer 2. Signer 2 is excluded from future sessions until he answers.*

2. ***Session 2***: *Signers* $= \{1, 3, \cdots, t+1\}$ *; The adversary quickly answers to escape Blocking2 state. At some point only a single Blocking2 signer is left on this session, let's assume that is signer 3. Signer 3 is excluded from future sessions until he answers.*

3. ***Session 3***: *Signers* $= \{1, 4, \cdots, t+2\}$ *; The adversary quickly answers to escape Blocking2 state. At some point only a single Blocking2 signer is left on this session, let's assume that is signer 4. Signer 4 is excluded from future sessions until he answers.*

   ⋮

$n-t+1$. ***Session n-t+1***: *Signers* $= \{1, n-t+2, \cdots, n\}$ *; The adversary quickly answers to escape Blocking2 state. At some point only a single Blocking2 signer is left on this session, let's assume that is signer* $n-t+2$*. Signer* $n-t+2$ *is excluded from future sessions until he answers.*

At this point, the only available signers in PotentialSigners are $\{1, n-t+3, \cdots, n\}$ *which is* $t-1$ *signers, so a new session cannot be started until one of the previous sessions i.e sid* $\in [n-t-3]$ *is unblocked (this will happen eventually). As soon as that is done however, the adversarial party will be added to Blocking3 state for that session and it will be impossible to escape (in reality since the procedure CheckNewSession() is run before the party gets added to Blocking3, the adversary will be able to join exactly one session at that instant before being added to Blocking3 state). As soon as he is added to Blocking3, the only thing he can do it to block all the current sessions in which he is, that is* $n-t+2$ *sessions in total.*
Generalizing, in the case where we have f *adversaries, each one of them will be able to block at most* $n-t+2$ *sessions before being forced into Blocking3 state and excluded from PotentialSigners.* ***This is the new invariant of the protocol*** *: an adversarial party can block at most* $n-t+2$ *sessions (we symbolize* $\ell = n-t+2$ *for clarity).*
We claim that the above scenario is a worst-case scenario as it is literally impossible (due to the construction) to create a new session for the adversary to block.
It is easy to see why that holds even when there are more than 1 *adversarial parties : one by one they will start being forced into Blocking3 states and be excluded from PotentialSigners.*

The proof as to why our construction works is very similar to the contradiction argument of the original ROAST: Let us assume that no internal signing session terminates. From observation 1, we know that we will have a constant stream of signing sessions. Since there are t honest signer present, by construction we will always (eventually) have t parties in the PotentialSigners set => a new session can always be started. As a result there will eventually by $f \cdot \ell + 1$ *sessions during the execution. By the invariant we know that the* f *adversarial parties are blocking in at most* ℓ *sessions each. Thus, among the* $f \cdot \ell + 1$ *sessions, there exists a session in which all signers are honest. This session is guaranteed to succeed and we've therefore reached a contradiction. Thus, the Sparkling ROAST protocol (parametrized with the specific PotentialSigners set of Equation 11.1) will always terminate in a signature so long as t honest parties are present.*

Finally, we've also proven that Sparkling ROAST(parametrized with the specific PotentialSigners set of Equation 11.1) needs at most $f \cdot \ell + 1$ *internal Sparkle sessions. Since*

we have t honest signers, it holds that $f \leq n - t$ and of course it must hold that $f \leq t - 1$. In order to achieve provable adaptive security of Sparkle in the ROM, we might have to assume $f \leq \frac{t-1}{2}$. Depending on what we want from the internal protocol, f can be bounded by different values all of which are all $< n$. Therefore a definite upper bound of internal sessions is $n \cdot \ell = n \cdot (n - t + 2) = O(n^2)$.

11.2.4 Using different *PotentialSigners* sets

The analysis of the Sparkling ROAST protocol depends heavily on the PotentialSigners sets with which it is parametrized. The reason why the adversaries can block a linear amount of sessions with the set of the previous section is because the set presented in Equation 11.1 does not limit the amount of sessions one can be in Pending state (this allows the adversary to join as many sessions as the latency allows him). As we briefly discussed earlier, the fact that we have two blocking points in the Sparkle version of ROAST, naturally draws us to the conclusion that the new protocol should allow adversaries to block at most 2 sessions. This would allows us to keep the total amount of internal sessions linear, just like in the case of the original ROAST. In order to do so, we can change the PotentialSigners set to the following:

$$PotentialSigners = \left\{ P_i \;\middle|\; \begin{array}{l} \neg\exists sid : P_i \in Blocking2[sid] \\ \wedge\ \neg\exists sid : P_i \in Blocking3[sid] \\ \wedge\ \neg\exists sid, sid' : sid \neq sid' \wedge P_i \in Pending[sid] \wedge P_i \in Pending[sid'] \\ \wedge\ Available\_PreSigs[P_i] \geq 1 \end{array} \right\} \tag{11.2}$$

It is easy to see (for example by using the previous scenario as an example), that if we were to use this version of PotentialSigners, an adversarial party would only be able to join at most 2 sessions before being forced into Blocking3 state. However, it is impossible to prove robustness with just t honest parties. The reason for that is that is that with this new PotentialSigners set, a new issue is introduced, the issue which we call hostage-taking.

This issue simply states that any honest participant who is stuck on Pending state for 2 sessions, is no longer able to join a new one. This completely breaks the notion of robustness as even 1 honest hostage is enough to stop the t honest total signers from producing a signature. However, since keeping the total amount of internal sessions linear is intriguing, we analyze how many honest participants need to be present for robustness with this new PotentialSigners set. To do so, we have to find the maximum amount of hostages depending on f. After doing so, we will need $|Honest| \geq |hostages| + t$ for robustness.

It is easy to see that the only way hostages can be created (and kept forever) is for the adversaries to stay at Blocking2 state forever. This way, it will be completely impossible for the honest signers to escape their Pending states. Let's consider than an adversary will hold a session of $t - 1$ honest participants at state Blocking2, let's then assume that these $t - 1$ honest signers will join a new session where a second adversary will hold at

state Blocking2. In the worst case, the $t-1$ signers will be all be added to the same two sessions each one blocked by a different Blocking2 adversary. Therefore, we can clearly see that each pair of 2 adversaries, can take $t-1$ hostages. Therefore, the total amount of hostages is : $|hostages| \leq \frac{f\cdot(t-1)}{2}$. This means that with this new PotentialSigners set, in order to guarantee robustness we will need $t + \frac{f\cdot(t-1)}{2}$ honest participants instead of the t of the previous section. The upside to this version of PotentialSigners is that the total amount of internal sessions will be $2 \cdot f + 1 = O(n)$ (linear to n in contrast to the previous section).

11.3 Extending Sparkling ROAST to other multi-round schemes

We argue that our construction of Sparkling ROAST, can be extended to other multi-round threshold signing schemes as long as they have the Identifiable Abort property. In such cases, all we have to do is use the piggybacking approach once again and add Blocking states for each blocking point of the protocol i.e if the original internal protocol is 5 rounds, like Glacius [29], there should be 4 blocking states. After each Blocking state, there should be a semi-responsive Pending state so that honest parties are not blocked by adversaries on Blocking states. We believe the analysis to be very similar as the one presented here.

11.4 A note on unforgeability of Sparking ROAST

Just like the original ROAST bases its unforgeability on the unforgeability of FROST [54], we base the unforgeability of Sparkling ROAST on the unforgeability of Sparkle+[28]. In fact, we believe that our wrapper protocol should even work in the case of an adaptive adversary.

Chapter 12

Conclusion and future work

Throughout this thesis we studied multi- and threshold Schnorr signatures looking at schemes, comparing them, looking at different threats and attacks, listing applications for different schemes and finally presenting a new robustness protocol for Identifiable Abort schemes.

Overall, threshold and multi- signatures are clearly a very hot research area. The fact that NIST [11] has made calls on threshold signature schemes proves that such schemes are desired (and needed) in order to add further security guarantees and make the internet more secure. Whether or not threshold signature schemes will be accepted for real-world use cases however depends on if they have desired properties such as:

- *Adaptive security.*
- *Robustness.*
- *Low amount of rounds.*
- *Low communication costs.*
- *Low computational costs.*
- *Signature size relative to that of single-party schemes.*

With the new papers on adaptive security [58], [57], it seems that researchers of the field are mainly focused into providing schemes that follow the instructions and standards set by NIST in their public call.

However, threshold signatures schemes are not limited to the standardization efforts of NIST. There are works presenting different threshold signature schemes (not necessarily Schnorr) that provide a plethora of properties which might be appealing for future applications. Examples include [59] where signers can append their partial signature share to an already produced threshold signature, effectively increasing the threshold, [60] where a threshold signing scheme is presented which supports private accountability i.e a selected trusted party can see which members were included in the signing process of a signature and a lot more. It seems that this research area can be extended as far as the creativity of the researchers involved.

As a future work, we are unaware of the existence any blind Schnorr threshold signing schemes. The fact that the ROS attack [15] broke many blind Schnorr constructions, added

a whole in the literature in the area of blind Schnorr. This is a big challenge as the ROS attacks do not only affect blind Schnorr but also threshold signature schemes, so creating a threshold blind Schnorr construction would have to do so by avoiding both threats.

Apart from that, it's interesting to see how threshold signature schemes can be extended to hold other properties, for example threshold ring signatures [61]. These can be extended further to add other properties such as designated verifiers [62], [63]. Finally, we mentioned that applications of threshold signatures include electronic voting so another interesting idea would be to use threshold (blind) signatures in voting protocols to implement registration authorities in a distributed manner, an addition to those schemes would of course be everlasting privacy and coercion resistance [64], [65].

Βιβλιογραφία

[1] Adi Shamir. How to Share a Secret. Communications of the ACM, 22(11):612–613, 1979.

[2] Paul Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. Proceedings of the 28th Annual Symposium on Foundations of Computer Science (SFCS '87), σελίδες 427–438. IEEE, 1987.

[3] T. P. Pedersen. Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing. Advances in Cryptology - CRYPTO '91Joan Feigenbaum, επιμελητής, τόμος 576 στο Lecture Notes in Computer Science, σελίδες 129–140. Springer, 1992. Presented at CRYPTO '91.

[4] Mihir Bellare και Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications SecurityDorothy E. Denning, Ravi Pyle, Ravi Ganesan, Ravi S. Sandhu και Victoria Ashby, επιμελητές, σελίδες 62–73, Fairfax, Virginia, USA, 1993. ACM.

[5] Georg Fuchsbauer, Eike Kiltz και Julian Loss. The Algebraic Group Model and its Applications. CRYPTO 2018Hovav Shacham και Alexandra Boldyreva, επιμελητές, τόμος 10992 στο Lecture Notes in Computer Science, σελίδες 33–62, Santa Barbara, CA, USA, 2018. Springer.

[6] Claus Schnorr. Efficient Signature Generation by Smart Cards. Journal of Cryptology, 4(3):161–174, 1991.

[7] Claus P. Schnorr. Efficient Identification and Signatures for Smart Cards. Advances in Cryptology - CRYPTO '89, τόμος 435 στο Lecture Notes in Computer Science, σελίδες 239–252. Springer, 1990.

[8] Amos Fiat και Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. Advances in Cryptology — CRYPTO '86Andrew M. Odlyzko, επιμελητής, τόμος 263 στο Lecture Notes in Computer Science, σελίδες 186–194. Springer, 1987.

[9] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[10] Luís T. A. N. Brandão και Michael Davidson. *Notes on Threshold EdDSA/Schnorr Signatures*. NIST Internal Report 8214B, National Institute of Standards and Technology, 2022. Initial Public Draft.

[11] Luís T. A. N. Brandão και René Peralta. *NIST First Call for Multi-Party Threshold Schemes*. NIST Internal Report 8214‟, National Institute of Standards and Technology, 2025. Second Public Draft.

[12] Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro και Chenzhi Zhu. *Snowblind: A Threshold Blind Signature in Pairing-Free Groups*. Cryptology ePrint Archive, Paper 2023/1228, 2023.

[13] Claus Peter Schnorr. *Security of Blind Discrete Log Signatures against Interactive Attacks*. Information and Communications Security (ICICS 2001)S. Qing, T. Okamoto και J. Zhou, επιμελητές, τόμος 2229 στο Lecture Notes in Computer Science, σελίδες 1–12. Springer, 2001.

[14] David Wagner. *A Generalized Birthday Problem*. Advances in Cryptology — CRYPTO 2002Moti Yung, επιμελητής, τόμος 2442 στο Lecture Notes in Computer Science, σελίδες 288–303, Heidelberg, 2002. Springer.

[15] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù και Mariana Raykova. *On the (in)security of ROS*. Cryptology ePrint Archive, Paper 2020/945, 2020.

[16] Gregory Maxwell, Andrew Poelstra, Yannick Seurin και Pieter Wuille. *Simple Schnorr multi-signatures with applications to Bitcoin*. Cryptology ePrint Archive, Report 2018/068, 2018. Version 20180118:124757.

[17] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven και Igors Stepanovs. *On the Security of Two-Round Multi-Signatures*. Cryptology ePrint Archive, Paper 2018/417, 2018.

[18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin και Pieter Wuille. *Simple Schnorr Multi-Signatures with Applications to Bitcoin*. Cryptology ePrint Archive, Paper 2018/068, 2018.

[19] Jonas Nick, Tim Ruffing και Yannick Seurin. *MuSig2: Simple Two-Round Schnorr Multi-signatures*. Advances in Cryptology — CRYPTO 2021Tal Malkin και Chris Peikert, επιμελητές, τόμος 12825 στο Lecture Notes in Computer Science, σελίδες 189–221. Springer, 2021. Virtual Event, August 16–20, 2021.

[20] Torben Pryds Pedersen. *A Threshold Cryptosystem without a Trusted Party*. Advances in Cryptology — EUROCRYPT ’91Donald W. Davies, επιμελητής, τόμος 547 στο Lecture Notes in Computer Science, σελίδες 522–526, Berlin, Heidelberg, 1991. Springer.

[21] Chelsea Komlo και Ian Goldberg. *FROST: Flexible Round-Optimized Schnorr Threshold Signatures*. Cryptology ePrint Archive, Paper 2020/852, 2020.

[22] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk και Tal Rabin. *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*. J. Cryptology, 20:51-83, 2007.

[23] Chelsea Komlo, Ian Goldberg και Douglas Stebila. *A Formal Treatment of Distributed Key Generation, and New Constructions*. Cryptology ePrint Archive, Paper 2023/292, 2023.

[24] Mihir Bellare, Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro και Chenzhi Zhu. *Better than Advertised Security for Non-Interactive Threshold Signatures*. Springer-Verlag, 2022.

[25] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch και Dominique Schröder. *ROAST: Robust Asynchronous Schnorr Threshold Signatures*. Cryptology ePrint Archive, Paper 2022/550, 2022.

[26] Mihir Bellare, Stefano Tessaro και Chenzhi Zhu. *Stronger Security for Non-Interactive Threshold Signatures: BLS and FROST*. Cryptology ePrint Archive, Paper 2022/833, 2022.

[27] Sela Navot και Stefano Tessaro. *One-More Unforgeability for Multi- and Threshold Signatures*. Cryptology ePrint Archive, Paper 2024/1947, 2024.

[28] Elizabeth Crites, Chelsea Komlo και Mary Maller. *Fully Adaptive Schnorr Threshold Signatures*. Cryptology ePrint Archive, Paper 2023/445, 2023.

[29] Renas Bacho, Sourav Das, Julian Loss και Ling Ren. *Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security*. Cryptology ePrint Archive, Paper 2024/1628, 2024.

[30] M. Bellare και G. Neven. *Multi-signatures in the plain public-key model and a general forking lemma*. Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)A. Juels, R. N. Wright και S. D. C.di Vimercati, επιμελητές, CCS '06, σελίδες 390-399. ACM, 2006.

[31] Shay Gueron, Simon Johnson και Jesse Walker. *SHA-512/256*. Cryptology ePrint Archive, Paper 2010/548, 2010.

[32] Shafi Goldwasser, Silvio Micali και Ronald L. Rivest. *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*. SIAM Journal on Computing, 17(2):281-308, 1988.

[33] David Pointcheval και Jacques Stern. *Security Arguments for Digital Signatures and Blind Signatures*. Journal of Cryptology, 13(3):361-396, 2000.

[34] K. Itakura και K. Nakamura. *A public-key cryptosystem suitable for digital multisignatures*. NEC Research and Development, 71:1-8, 1983.

[35] Accredited Standards Committee X9. *American National Standard X9.62-2005, Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

[36] Pieter Wuille, Jonas Nick και Tim Ruffing. *BIP-340: Schnorr Signatures for secp256k1*. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki, 2020. Accessed: 2025-06-02.

[37] Chrysa Stathakopoulou και Christian Cachin. *Threshold Signatures for Blockchain Systems*. Research Report ZYP1704-014, Swiss Federal Institute of Technology, IBM Research - Zurich, 2017.

[38] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A. Kroll, Edward W. Felten και Arvind Narayanan. *Securing Bitcoin Wallets via a New DSA/ECDSA Threshold Signature Scheme*. Technical Report, Princeton University, 2015.

[39] Frostsnap. *Introducing Frostsnap*. https://frostsnap.com/introducing-frostsnap.html.

[40] Christian Cachin. *Distributing Trust on the Internet*. Proceedings of the International Conference on Dependable Systems and Networks (DSN 2001), σελίδες 183-192, Göteborg, Sweden, 2001. IEEE Computer Society.

[41] Lidong Zhou, Fred B. Schneider και Robbertvan Renesse. *COCA: A Secure Distributed Online Certification Authority*. ACM Transactions on Computer Systems (TOCS), 20(4):329-368, 2002.

[42] Christian Cachin και Asokan Samar. *Secure Distributed DNS*. Proceedings of the International Conference on Dependable Systems and Networks (DSN 2004), σελίδες 423-432, Florence, Italy, 2004. IEEE Computer Society.

[43] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer και Bryan Ford. *Scalable Bias-Resistant Distributed Randomness*. 2017 IEEE Symposium on Security and Privacy (SP), σελίδες 444-460. IEEE Computer Society Press, 2017.

[44] Abida Haque, Stephan Krenn, Daniel Slamanig και Christoph Striecks. *Logarithmic-Size (Linkable) Threshold Ring Signatures in the Plain Model*. Public Key Cryptography - PKC 2022, Part II, τόμος 13178 στο Lecture Notes in Computer Science, σελίδες 437-467. Springer, 2022.

[45] Georg Fuchsbauer, Antoine Plouviez και Yannick Seurin. *Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model*. Cryptology ePrint Archive, Paper 2019/877, 2019.

[46] Toru Okamoto. *Provably Secure Identification Schemes and Corresponding Signature Schemes*. Advances in Cryptology — CRYPTO '92Ernest F. Brickell, επιμελητής, τόμος 740 στο Lecture Notes in Computer Science, σελίδες 31-53. Springer, 1993. Presented at CRYPTO '92.

[47] Mihir Bellare, Alexandra Boldyreva και Jessica Staddon. *Randomness Re-use in Multi-recipient Encryption Schemes*. Public Key Cryptography (PKC), σελίδες 85-99, 2003.

[48] Kannan Balasubramanian. *Security of the secp256k1 Elliptic Curve Used in the Bitcoin Blockchain*. Indian Journal of Cryptography and Network Security, 4(1):12–16, 2024.

[49] Thomas Ristenpart και Scott Yilek. *The Power of Proofs-of-Possession: Securing Multi-party Signatures against Rogue-Key Attacks*. Advances in Cryptology – EUROCRYPT 2007Moni Naor, επιμελητής, τόμος 4515 στο Lecture Notes in Computer Science, σελίδες 228–245, Barcelona, Spain, 2007. Springer.

[50] Douglas R. Stinson και Reto Strobl. *Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates*. Proceedings of the 6th Australasian Conference on Information Security and Privacy (ACISP 2001), σελίδες 417–434, 2001.

[51] H. W. H. Wong, J. P. K. Ma, H. H. F. Yin και S. S. M. Chow. *Real Threshold ECDSA*. Proceedings of the Network and Distributed System Security Symposium (NDSS), σελίδες 1–18, 2023.

[52] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos Adrian Seredinschi, Orr Tamir και Alin Tomescu. *SBFT: A Scalable and Decentralized Trust Infrastructure*. Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), σελίδες 568–580, 2019.

[53] Yizhong Liu, Andi Liu, Yuan Lu, Zhuocheng Pan, Yinuo Li, Jianwei Liu, Song Bian και Mauro Conti. *Kronos: A Secure and Generic Sharding Blockchain Consensus with Optimized Overhead*. Cryptology ePrint Archive, Paper 2024/206, 2024.

[54] Elizabeth Crites, Chelsea Komlo και Mary Maller. *How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures*. Cryptology ePrint Archive, Paper 2021/1375, 2021.

[55] Elizabeth Crites, Chelsea Komlo και Mary Maller. *Fully Adaptive Schnorr Threshold Signatures*. Cryptology ePrint Archive, Paper 2023/445, Version 20230327:090418, 2023. Version 20230327:090418.

[56] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner και Chenzhi Zhu. *Twinkle: Threshold Signatures from DDH with Full Adaptive Security*. Cryptology ePrint Archive, Paper 2023/1482, 2023.

[57] Elizabeth Crites, Chelsea Komlo και Mary Maller. *On the Adaptive Security of Key-Unique Threshold Signatures*. Cryptology ePrint Archive, Paper 2025/943, 2025.

[58] Elizabeth Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro και Chenzhi Zhu. *On the Adaptive Security of FROST*. Springer-Verlag, 2025.

[59] Diego F. Aranha, Mathias Hall-Andersen, Anca Nitulescu, Elena Pagnin και Sophia Yakoubov. *Count Me In! Extendability for Threshold Ring Signatures*. Cryptology ePrint Archive, Paper 2021/1240, 2021.

[60] Dan Boneh και Chelsea Komlo. *Threshold Signatures with Private Accountability*. Cryptology ePrint Archive, Paper 2022/1636, 2022.

[61] Abida Haque και Alessandra Scafuro. *Threshold Ring Signatures: New Definitions and Post-Quantum Security*. Cryptology ePrint Archive, Paper 2020/135, 2020.

[62] Pourandokht Behrouz, Panagiotis Grontas, Vangelis Konstantakatos, Aris Pagourtzis και Marianna Spyrakou. *Designated-Verifier Linkable Ring Signatures*. Cryptology ePrint Archive, Paper 2022/470, 2022.

[63] Danai Balla, Pourandokht Behrouz, Panagiotis Grontas, Aris Pagourtzis, Marianna Spyrakou και Giannis Vrettos. *Designated-Verifier Linkable Ring Signatures with unconditional anonymity*. Cryptology ePrint Archive, Paper 2022/1138, 2022.

[64] Panagiotis Grontas, Aris Pagourtzis και Marianna Spyrakou. *Voting with coercion resistance and everlasting privacy using linkable ring signatures*. Cryptology ePrint Archive, Paper 2025/002, 2025.

[65] Panagiotis Grontas και Aris Pagourtzis. *Anonymity and everlasting privacy in electronic voting*. International Journal of Information Security, 22(4):819–832, 2023.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

ROM	*Random Oracle Model*
AGM	*Algrebraic Group Model*
PPT	*Probabilistic Polynomial Time*
DLP	*Discrete Logarithm Problem*
DLOG	*Discrete Logarithm Assumption*
(A)OMDL	*(Algebraic) One-More Discrete Logarithm*
DDH	*Decisional Diffie Hellman*
ROS	*Random inhomogeneities in a Overdetermined Solvable system of linear equations*
DKG	*Distributed Key Generation*
SSS	*Shamir's Secret Sharing*
VSS	*Verifiable Secret Sharing*
PVSS	*Pedersen's Verifiable Secret Sharing*
AgVSS	*Aggregatable Verifiable Secret Sharing*
NIKE	*Non-Interactive Key Exchange*
ROAST	*Robust Asynchronous Schnorr Threshold Signatures*