2023

An Analysis of a Linear Algebra Based Group Key Exchange Protocol

Annie Zhang
University of Vermont

An Analysis of a Linear Algebra Based Group Key Exchange Protocol

A Thesis Presented

by

Annie Zhang

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Mathematical Sciences

August, 2023

Defense Date: May 24, 2023
Thesis Examination Committee:

Christelle Vincent, Ph.D, Advisor
Joe Near, Ph.D, Chairperson
Taylor Dupuy, Ph.D
Cynthia J. Forehand, Ph.D, Dean of the Graduate College

Abstract

Group key exchange protocols are used to establish session keys, which can then be used as encryption keys to set up secure channels of communication, between more than two parties simultaneously. Many different group key exchange protocols exist and require security proofs in order to determine the strength of the protocol and answer the following questions: does the protocol provide authentication, and if so, to what degree? Does the protocol provide key secrecy? In this thesis we examine a particular group key exchange protocol that we call the *vector space projection protocol* as first described in "A Group Key Establishment Scheme" by Guzey, Kurt, and Ozdemir, and show using a particular type of security proof — the game based security model — that the protocol as described does not achieve key secrecy. We show that there are at least four adversaries with non-negligible probabilities of winning the key secrecy security game, which indicates that this key exchange protocol is not one that should be implemented.

Dedication

To Dad: thank you for always having faith in me. I love you.

Acknowledgements

There are so many wonderful people to thank!

First to my advisor Christelle Vincent, whose patience, kindness, and sense of humor made working with her an extremely enjoyable experience. I went through one of the most difficult periods of my life while writing this thesis – the death of my father, a bout of COVID and other health problems, the breakdown of relationships with many people who were family – and were it not for her unfailing compassion, I definitely would never have finished this thesis.

Second to the rest of my committee, Joe Near and Taylor Dupuy, for their guidance, advice, and jokes.

Third to my partner Alan, who spent probably a hundred hours listening to me talk about this thesis, came up with some of the results in this thesis, and read every draft. You are the greatest teammate, partner, and best friend I could ever ask for.

Fourth to my wonderful friends: Tobias for being a fellow Iowan and having the same sense of humor as me, Gabe for being a fellow Iowan and overall the greatest person ever, Farrah for lovingly psychoanalyzing me and listening to me talk about math despite not even liking math, and all the rest for their kindness and jokes. I am so lucky to have befriended you all.

And lastly, to my dad. He was my biggest cheerleader, had an extremely large amount of faith in my ability to make smart decisions, always listened to me ramble about math, and gamely went ice skating with me on a regular basis despite not knowing how to ice skate. I love you, I miss you, and I hope I can show you this thesis in whatever life comes after this one.

Table of Contents

Chapter 1

Introduction

1.1 Outline of Thesis

In this thesis we discuss a particular group key exchange protocol, which we call the *vector space projection protocol*, as computation of the session key requires group members to project into vector spaces over finite fields, first described in [GKO21] as *Algorithm 2* and presented with a modification in their follow up work [COG21]. We then play the key secrecy security game to determine whether or not the protocol still provides key secrecy in the face of adversarial interaction. For the key secrecy game, we follow the general outline for security games outlined in [dSGFW20] but use the definition of the adversary's advantage in the key secrecy game from [BCP01].

A group key exchange protocol is used to establish a shared secret, also known as a session key, between more than two users at a time. This shared secret can then be used to authenticate users and establish secure channels of communication between them [Har13]. These protocols are of interest due to their potential for reducing the computational power needed to establish shared secrets between groups of people, as

multiple users can now simultaneously derive the same session key instead of having to sequentially derive it, one user after another.

In [GKO21], the vector space projection protocol relies on picking a vector space over a finite field, $\mathbb{F}_q^n$, and a secret subspace W. From there, the group manager releases a vector $\vec{v} \notin W$ that is also nonzero and each remaining group member projects $\vec{v}$ into W. However, since $\mathbb{F}_q$ is a finite field, usual orthogonal projection will not work, so we propose a suitable definition of projection into a subspace over a finite field for this protocol. A literature search revealed that there are alternate definitions of projection over finite fields – see [Che18] for two such definitions – but we didn't find any definitions that were not computationally intensive.

For any group key exchange protocol it is important that we determine whether or not the protocol can provide key secrecy, or in other words, that all adversaries have a negligible advantage when trying to guess or compute the shared secret. If just one adversary has a non-negligible advantage, then the protocol doesn't provide key secrecy. To be precise, we are interested in computing for each adversary exactly how much she can increase her chance of correctly guessing or even computing the shared secret by gathering information about the protocol itself. This information gathering is a key component of the key secrecy game, which we play once against each adversary: we simulate data leaks by allowing the adversary to ask for as many user's private keys and past session keys as she wants, and then she'll test herself by trying to correctly distinguish between an actual, unrevealed session key and a random element from the key distribution space. We play the key secrecy game with 4 different adversaries and show that all of them have a non-negligible chance of either correctly computing the session key or distinguishing between the actual key

and a random element, which leads us to recommend against using the vector space projection protocol. In fact, 3 of the 4 adversaries have a 100% probability of correctly computing the session key.

This thesis is outlined as follows. In Chapter 2, we will describe the vector space projection protocol from [GKO21] in detail and provide our definition of projection into subspaces over a finite field. Then in Chapter 3 we will describe security goals and security games in general before describing the key secrecy game, how it's played, and how to compute the adversary's advantage for this security game. In Chapter 4 we provide strategies for the adversary: we play the key secrecy game with 4 different adversaries and show that each adversary has a non-negligible advantage, which provides proof that the key exchange protocol does not provide key secrecy.

1.2 Related Works

This thesis discusses group key exchange protocols, which differ from group key *agreement* protocols in that the former requires a group manager while the latter does not have a group manager. Group key agreement protocols are also well studied; in one of the first proposed schemes, Ingemarsson et al. extend Diffie-Hellman to a multi-user case [ITW82].

A seminal work in the formal study of group key exchange protocols is the work by Bresson et al. [BCPQ02], which provides a formal security model and definitions for *static* group key exchanges. Later in [BCP01] Bresson et al. they provide a formal security model and definitions for *dynamic* group key exchanges and this latter model provides part of the framework we use to analyze the vector space projection proto-

col. In [BCP02a], Bresson et al. extend their work on the dynamic case to include additional components, for example allowing for concurrent executions of the protocol and additional modes of corruption by an adversary. Finally in [BCP02b] the authors present a formal treatment to determine whether or not a group Diffie-Hellman key exchange is secure against dictionary attacks.

One assumption that Bresson et al. make in [BCP02a], [BCP02b], [BCP01], and [BCPQ02] when constructing their security models is that none of the participants in the group key exchange protocols behave maliciously. Everyone except for the adversary is following the protocol correctly. In response, Bohli et al. [BGVS07] extend the formal security models from these earlier papers by giving a formal definition of the security properties addressing malicious participants and provide a proof that shows, under this extended security model, that a variant of the group key exchange protocol introduced by Kim et al. [KLL04] is secure. Katz and Shin [KS05] discovered that Bresson et al.'s models insufficiently addressed impersonation attacks. Gorantla et al. [GBNM08] investigated the security of group key exchange protocols under the key compromise impersonation attack specifically.

With regards to the definitions of important concepts in game based security models, like *authentication* and *secrecy*, as well as how these definitions relate to each other, [dSGFW20] provide clear, predicate-based definitions and explain how these concepts relate to each other. They build on work first done by Bellare and Rogaway [BR94] in formalizing certain security notions. There have been many other papers since then that have provided modifications and variations to Bellare and Rogaway's work, including for example extensions of adversarial powers [CK01] and formalizations of key confirmation properties [FGSW16]. In [dSGFW20] in particular,

their framework for security games comes from [BFWW11] with augmentations to the framework from [FGSW16].

Finally in [PRSS21], the authors give an overview of several papers that develop group key exchange protocols with respect to a formal computational game based security model in order to compare and analyze said models. We invite the interested reader to look there for additional information on many of the different group key exchange protocols that have been proposed over the years and the security models constructed and used to analyze them.

Chapter 2

The Vector Space Projection Protocol

In this chapter we'll describe the group key exchange protocol outlined in [GKO21], which we will call the *vector space projection protocol*. A *group key exchange protocol*, which we denote $\boldsymbol{\pi}$, is a key exchange protocol designed to generate and share a session key with a group of individuals. The vector space projection protocol is meant to be *dynamic*, which according to [BCP01] means the protocol should allow for changes in the compositions of the group members without compromising security. To be more precise, let $\mathcal{U}$ be the entire set of users who can participate in the protocol and $\mathcal{I} \subset \mathcal{U}$ the subset of users who are currently participating. Then users can be added to $\mathcal{I}$ from $\mathcal{U}$, while any user who could be removed would then join $\mathcal{U} \setminus \mathcal{I}$. We will describe the algorithm in the vector space projection protocol that allows for users to be added later in this section; the protocol does not actually provide an algorithm to remove users. The group of users $\mathcal{I}$ participating in the protocol has what we call a *group manager*, denoted GM, who is responsible for setting up the key exchange protocol.

To be more precise, the group manager will run the **SETUP** algorithm to set up the key exchange protocol. The group manager begins by selecting a finite field of size q and a positive integer n. Next the group manager generates a random basis $\mathcal{F} = \{\vec{v}_1, \ldots, \vec{v}_n\}$ for $\mathbb{F}_q^n$ and lets W be the subspace generated by the span of $\mathcal{B} = \{\vec{v}_1, \ldots, \vec{v}_r\}$, where $r < n$, and keeps W a secret. Then they pick a polynomial $f(x) \in \mathbb{F}_q[x]$ of degree d where $d > |\mathcal{U}|$. The two security parameters are q and d. Finally, they pick $r-1$ random nonzero elements of $\mathbb{F}_q$ that we denote $n_1, \ldots, n_{r-1}$.

The group manager keeps the basis $\mathcal{F}$, W and its basis $\mathcal{B}$, $f(x)$, and the $r-1$ random elements $n_1, \ldots, n_{r-1}$ secret. Each remaining user in the group is given their own unique point $(x_i, f(x_i))$ on the graph of the polynomial $f(x)$; x_i is user i's public key. The remaining group members are also all given a unique basis $\mathcal{F}_i$ for $\mathbb{F}_q^n$. To generate group member i's basis the group manager first rescales every vector in $\mathcal{B}$ in the following manner to generate a basis $\mathcal{B}_i$ for W:

$$\{f(x_i)\vec{v}_1, n_1 f(x_i)\vec{v}_2, n_2 f(x_i)\vec{v}_3, \ldots, n_{r-1} f(x_i)\vec{v}_r\}.$$

In other words they multiply each vector in $\mathcal{B}$ by a different scalar. Because $f(x_i)$ is different for each group member, each group member's basis $\mathcal{B}_i$ will be unique, though they are all multiples of each other. Group member i's unique basis $\mathcal{F}_i$ for $\mathbb{F}_q^n$ will be $\mathcal{B}_i \cup \{v_{r+1}, \ldots, \vec{v}_n\}$:

$$\{f(x_i)\vec{v}_1, n_1 f(x_i)\vec{v}_2, n_2 f(x_i)\vec{v}_3, \ldots, n_{r-1} f(x_i)\vec{v}_r, \vec{v}_{r+1}, \ldots, \vec{v}_n\}. \tag{2.1}$$

User i's basis $\mathcal{F}_i$ is their private key.

In summary: the group manager keeps the original vector space, the subspace W

and its basis $\mathcal{B}$, the function $f(x)$, and the $r-1$ random integers secret while publicly revealing the dimension r of W and the values x_i that are inputs for $f(x)$. Each of the remaining group members' public key will be x_i while their private key will be their unique basis $\mathcal{F}_i$.

To add users from $\mathcal{U} \setminus \mathcal{I}$ we have the **JOIN** protocol, the algorithm that a user (who *isn't* the group manager) can initiate in order to add someone new to the group. Suppose that user i wants to add user j. First user i will pick a random element $s \in \mathbb{F}_q^\times$ and will multiply every vector in their basis $\mathcal{F}_i$ by s in order to generate a new basis $\mathcal{F}_j$:

$$\{sf(x_i)\vec{v}_1, sn_1f(x_i)\vec{v}_2, sn_2f(x_i)\vec{v}_3, \ldots, sn_{r-1}f(x_i)\vec{v}_r, s\vec{v}_{r+1}, \ldots, s\vec{v}_n\}$$

Now user j has their own unique basis, $\mathcal{F}_j$, for $\mathbb{F}_q^n$. This basis is their private key; their public key will be the ordered pair (x_i, j).

Now we can describe the vector space projection protocol, known as *Algorithm 2* in [GKO21]. Whenever a session key is needed, the group manager will publicly share a nonzero vector $\vec{v} \notin W$ and the remaining group members will project $\vec{v}$ onto W. The projection, which we will denote $\vec{u}$, is the session key. Each time a new session key is needed, the group manager will release a new vector.

2.1 Projecting onto Subspaces Over a Finite Field

In this section we present a technique for projection onto a subspace over a finite field. We did not find a simple description in the literature of projecting a vector onto a subspace over a finite field, though definitions that require more computational overhead than is ideal for a key exchange protocol have been proposed (see [Che18] for two such definitions).

The vector space projection protocol rests on being able to project some vector $\vec{v}$ onto the secret subspace W. Since there's no inner product in the larger vector space $\mathbb{F}_q^n$, the group manager needs to define a projection $\mathcal{P}$ onto W as we cannot simply use the orthogonal projection formula.

Since one of the goals of a group key exchange protocol is that derivation of the session key is not computationally intensive, we suggest the following as a definition for projection over a finite field. Let $\mathcal{F} = \{\vec{v}_1, \ldots, \vec{v}_n\}$ be a basis for $\mathbb{F}_q^n$. Let W be the span of the first r basis vectors. We define *projection onto W via $\mathcal{F}$* as follows: if we're projecting $\vec{v}$ we first write $\vec{v}$ as

$$\vec{v} = a_1\vec{v}_1 + \cdots + a_n\vec{v}_n.$$

When $\vec{v}$ is written in this form, to project onto W we simply set $a_i = 0$ if $\vec{v}_i$ is not one of the r basis vectors for W and leave it untouched otherwise. In other words,

the projection $\vec{u}$ is

$$\vec{u} = a_1\vec{v}_1 + \cdots + a_r\vec{v}_r.$$

Then whenever a session key is needed, the group manager will share a vector $\vec{v}$ and group member i will project $\vec{v}$ onto W via $\mathcal{F}_i$. Even though each group member has a different basis, they will all still end up with the same vector $\vec{u}$.

Proposition. *Let $\mathcal{F}_i$ and $\mathcal{F}_j$ be as in (2.1) and let $\vec{v} \notin W$ be nonzero. Then the projection of $\vec{v}$ onto W via $\mathcal{F}_i$ and the projection of $\vec{v}$ onto W via $\mathcal{F}_j$ yield the same vector.*

Proof. To begin, we'll write $\vec{v}$ as a linear combination of the vectors in $\mathcal{F}_i$ and $\mathcal{F}_j$:

$$\vec{v} = a_1 f(x_i)\vec{v}_1 + a_2 n_1 f(x_i)\vec{v}_2 + \cdots + a_r n_{r-1} f(x_i)\vec{v}_r + a_{r+1}\vec{v}_{r+1} + \cdots + a_n\vec{v}_n$$
$$\vec{v} = b_1 f(x_j)\vec{v}_1 + b_2 n_1 f(x_j)\vec{v}_2 + \cdots + b_r n_{r-1} f(x_j)\vec{v}_r + b_{r+1}\vec{v}_{r+1} + \cdots + b_n\vec{v}_n$$

Since

$$a_1 f(x_i)\vec{v}_1 + a_2 n_1 f(x_i)\vec{v}_2 + \cdots + a_r n_{r-1} f(x_i)\vec{v}_r + a_{r+1}\vec{v}_{r+1} + \cdots + a_n\vec{v}_n =$$
$$b_1 f(x_j)\vec{v}_1 + b_2 n_1 f(x_j)\vec{v}_2 + \cdots + b_r n_{r-1} f(x_j)\vec{v}_r + b_{r+1}\vec{v}_{r+1} + \cdots + b_n\vec{v}_n$$

and $\{\vec{v}_1, \ldots, \vec{v}_n\}$ forms a basis for $\mathbb{F}_q^n$ we know that

$$a_1 f(x_i) = b_1 f(x_j), a_2 n_2 f(x_i) = b_2 n_2 f(x_j), \ldots, a_r n_r f(x_j) = b_r n_r f(x_j),$$
$$a_{r+1} = b_{r+1}, \ldots, a_n = b_n$$

which tells us that

$$a_1 f(x_i)\vec{v}_1 + a_2 n_1 f(x_i)\vec{v}_2 + \cdots + a_r n_{r-1} f(x_i)\vec{v}_r =$$
$$b_1 f(x_j)\vec{v}_1 + b_2 n_1 f(x_j)\vec{v}_2 + \cdots + b_r n_{r-1} f(x_j)\vec{v}_r$$

□

Whenever a session key is needed, the group manager should publicly release a vector $\vec{v}$ written in terms of the standard basis for $\mathbb{F}_q^n$. Each remaining group member will then rewrite it in terms of their unique basis for W, project $\vec{v}$ into W in the manner described above, and then rewrite $\vec{u}$ in terms of the standard basis so that everyone will have the same session key.

Chapter 3

Security Games

For any key exchange protocol, one security goal that is commonly expected is *key secrecy*; that is, any active adversary isn't able to learn the session key [dSGFW20]. We note that this goal is separate from *authentication*, where only the "right" parties are allowed to participate in the protocol and derive the session key. A protocol can provide key secrecy without providing authentication; the original formulation of Diffie-Hellman, for example, does not provide any level of authentication because neither party provides any type of authentication for their public key, but key secrecy is guaranteed by the hardness of the computational Diffie-Hellman problem [DH76]. For the sake of completeness, we note that there are many different security goals that all center around authentication in some capacity: implicit and explicit key authentication, implicit and explicit key authentication, and key confirmation. We exclusively focus on key secrecy in this paper, so we will not dive in to the details of these other security goals.

The rest of this chapter is outlined as follows: we will first describe security games, the framework used to determine whether or not a given key exchange protocol

reaches a particular security goal. Then we will give a precise definition of the key secrecy security game, the game we play to determine whether or not the vector space projection protocol provides key secrecy.

The framework we present here is from [dSGFW20]. The framework consists of playing a security game to determine whether or not a given group key exchange protocol reaches a particular security goal. The players of the game are the individuals in $\mathcal{I}$ who can engage in the protocol and are running it to generate session keys, while an outside adversary is trying to trigger some "bad event", which we'll define in Section 3.5. We note that in the formulation of a security game, outside adversaries are not considered players; only the users engaging in the protocol are. We want to compute the probability of an adversary triggering that so-called "bad event". An *adversary* is a probabilistic polynomial-time algorithm that sends out messages in order to disrupt the game and trigger the "bad event". If the adversary is successful in doing so, we say that the adversary has won. As there could potentially be multiple such algorithms that can trigger the "bad event", we would refer to each separate algorithm as a separate adversary.

A protocol is said to meet its security goal if the advantage of any adversary is negligible as a function of the security parameter. An adversary's *advantage* is a function of her probability of winning the game; we will provide in Section 3.5 the exact function that is used to compute the adversary's advantage in the key secrecy game. We want her advantage to be *negligible* as a function of the security parameter q; a function $f(k)$ is negligible if for every $c > 0$ there exists $k_c > 0$ such that for all $k > k_c, f(k) < \frac{1}{k^c}$. In other words, $f(k)$ eventually decreases faster than any rational function (or its inverse increases faster than any polynomial function).

We can now describe how a security game is played. In a security game, the users will engage in the protocol as normal while an adversary will observe the interactions between users and send queries to obtain information about the protocol and the users. She sends her *queries*, the term we use to refer to any message the adversary sends or any request for information she makes, to a behavior algorithm χ that first determines if her queries are within the scope of the game and sends back the appropriate response if so. Later in Sections 3.4 and 3.5 we will more precisely define what it means to stay within the scope of security games in general and the key secrecy game in particular: it requires use of what we call a **Valid** predicate. These queries take on specific forms. Different formulations of the same security game will allow for different queries and there are many queries common across formulations. In both [BCP01] and [dSGFW20] there are the **Send, Reveal, Corrupt,** and **Test** queries. The **Send** query is the most general: the adversary can "send" whatever message she wants to any user (the behavior algorithm will respond the way the user would if her message stays within the scope of the game) or forward, alter, delay, or delete messages between users (by sending her request to the behavior algorithm who will carry it out provided it stays within the game's scope). She could, for instance, delete the public vector the GM has shared and send out her own vector instead. When the adversary sends the **Reveal** query regarding a specific session key (i.e. the current one or a past one) she is given the session key she requested (more precisely, the behavior algorithm will first check if the users have finished deriving that session key and give her the session key if so, as the adversary could send this query while the protocol is still running). We then say that this session has been *corrupted.* When the adversary sends the **Corrupt** query about a specific player she is given that player's

private key. We analogously call this player *corrupted.* We will explain the **Test** query when we discuss the key secrecy security game.

We allow the adversary to ask queries to model the real life scenario of information leaking and an adversary or adversaries using this information to determine past, current, or future session keys. In short, allowing for these data leaks lets us test the strength of our protocol: even if information has been leaked, can we still keep past, current, or future session keys a secret?

We'll spend the rest of this chapter giving a more precise definition of security games and explain how we keep track of queries, the responses to these queries, and how those responses affect the game. We follow the framework described in [BCP01] and [dSGFW20].

3.1 Protocols and Sessions

To be more precise, a key exchange protocol $\boldsymbol{\pi}$ consists of two algorithms **kg** and ζ. The first algorithm, which we usually write as $\mathbf{kg}(1^\lambda)$, where 1^λ is the unary representation of the security parameter λ, is a randomized key generation algorithm that gives everyone their public and private keys. The second algorithm ζ is the protocol that parties engage in to generate session keys.

In order to understand security games in more detail, we need to discuss sessions. For simplicity, we will first discuss the two party case; that is, when the group manager is only expecting one other person to compute the session key. When two parties are engaging in the protocol together, each party has their own *local session.* A local session is a temporary two-way link that enables communication between two parties

and has an owner, who we call the *session owner*, and an intended peer [dSGFW20]. As an example, if Alice and Bob are performing a Diffie-Hellman key exchange together, then Alice has a session where she's the owner and Bob the intended peer, while Bob has a session where he's the owner and Alice the intended peer. Since Alice and Bob are engaging in conversation together, their sessions should be identical in content once they've exchanged all the information needed to derive the session key. We say that the session has been *terminated* once they both stop talking to each other. In the case of a group key exchange protocol with more than two parties, the users engage in multiple sessions with other users simultaneously or in sequence [BCP01]. For the vector space projection protocol, the group manager would engage in multiple sessions simultaneously with all remaining users in $\mathcal{I}$.

To both identify and keep track of all the sessions that are created and terminated over the course of running a key exchange protocol, we maintain a list of valid *local session identifiers* $\ell \in \mathcal{U} \times \mathcal{U} \times \mathbb{Z}$ which is made public. Each local session identifier is of the form $\ell = (i, j, k)$ to indicate that this is the kth session between session owner i and their intended peer j. We will use $\ell.\mathbf{id}$ and $\ell.\mathbf{pid}$ to refer to a session's owner and intended peer respectively.

3.2 Protocol and Game States

Security games maintain a running list of both protocol and game states to keep track of the players' execution of the protocol and the adversary's actions taken over the course of the game. We mentioned the goal of the adversary is to trigger a "bad event" by sending queries while the players run the protocol and generate session

keys. Each time the adversary sends a query, information about the protocol might be leaked: for example, a **Reveal** query reveals the current session key, a **Corrupt** query reveals one or more players' private keys. As the game is played, we want to keep track of what's happened in each session: not only who the owner and intended peer are and whether or not a session key was derived, but also whether or not the owner or peer was corrupted and whether or not the adversary learned the session key for that session. In short, we need to keep track of what information about players and session keys has been leaked and when. To keep track, we maintain five running lists.

1. The list **LSID** of local session identifiers.
2. The list **SST** of protocol related session states.
3. The list **EST** of global protocol related information that's separate from session state information.
4. The list **LST** of game related local session states.
5. The list **MST** of global game related information that's separate from the game related local session states.

We'll now describe each of these lists. In the previous section, we discussed what local session identifiers are; the running list of those local session identifiers is **LSID**. Let $\{0,1\}^*$ be the set of all finite-length bit-strings.

The list **SST** of protocol related session states keeps track of what's happening during each session, providing answers to questions like "What is this session's session ID?", "Who is the session owner?", "Has this session terminated yet?". For each session $\ell = (i, j, k)$ its protocol related local session state consists of:

- $(\mathbf{pk}_i, \mathbf{sk}_i)$: the public and secret key of the session owner, player i.
- $\mathbf{pk}_j$: the public key of the intended peer, player j.
- **crypt** $\in \{0,1\}^*$: protocol-specific private session state used to maintain secret values from one invocation to the next.
- **accept** $\in \{\mathbf{true}, \mathbf{false}, \perp\}$: **accept** is initially set to $\perp$ to indicate the protocol is running. Once it's done, it changes to **true** or **false** to indicate whether or not the session owner accepted the session as a successful run of $\boldsymbol{\pi}$. This value is made public.
- **sid** $\in \{0,1\}^* \cup \{\perp\}$: **sid** is the session ID and is initially set to $\perp$ while the protocol is running. It will change to a non-trivial value if and only if **accept** is set to **true** (that is, if the session owner deemed the protocol run successful); otherwise it never changes. This value will change at most once per session and is also made public. The session ID is generated as a result of running the key exchange protocol and is made public. A session ID is used to keep track of the information that's been exchanged between the owner i and intended peer j. See Section 4.1 of [dSGFW20] for an example of what a session ID would look like for two parties running Diffie-Hellman.
- **key** $\in \{0,1\}^* \cup \{\perp\}$: **key** is the session key and is initially set to $\perp$. It will change if and only if **accept** is set to **true**, and will only change once per session (at most).
- **kcid** $\in \{0,1\}^* \cup \{\perp\}$: **kcid** is the key confirmation identifier, used to indicate which sessions will eventually derive the same session key. It's initially set to $\perp$

and can be changed once per session to a non-trivial value if and only if **key** is non-trivial. Like the session ID, the key confirmation identifier is made public.

- **kconf** $\in \{\textbf{full}, \textbf{almost}, \textbf{no}, \perp\}$: **kconf** indicates the type of key confirmation the session owner expects. It's initially set to $\perp$ and changed when the session is first activated. **Full** key confirmation means if session owner i engages in the protocol with intended peer j and that peer is following the protocol correctly, then there is a guarantee that someone else has derived the shared secret (this someone else could be the intended peer, but it might not be). **Almost full** key confirmation means if session owner i engages in the protocol with intended peer j and that peer is following the protocol correctly, then there is another session (and the owner of this other session might not be j) that has the same key confirmation identifier and *if* that other session has derived a key then it's the same one that session owner i has. The difference is that full confirmation guarantees the same session key; almost full guarantees the same key confirmation identifier. **No** key confirmation means the protocol does not guarantee that, if session owner i is engaging in the protocol honestly with intended peer j, that there will be someone else (intended peer j or otherwise) that has the same session key.

From now on, for a given session ℓ we will write ℓ.**accept** to refer to the value of **accept** for session ℓ, ℓ.**sid** refers to its session ID, and so on.

The idea behind the list **SST** is that as the players run the protocol and generate session keys, we update the protocol related session state for each session to keep track of what is currently happening in each session. It also allows us to keep track of what has happened in previous sessions as well.

Next we'll describe the components of the list **EST**, which contains global protocol related information. This list is a set of tuples that we denote as $\mathcal{L}_{\mathbf{keys}}$ that keeps track of all players and tells us which of them have been corrupted: $\mathcal{L}_{\mathbf{keys}} = \{(i, \mathbf{pk}_i, \mathbf{sk}_i, \delta_i) : i \in \mathcal{U}\}$ where $\delta_i \in \{\mathbf{honest}, \mathbf{corrupt}\}$ tells us whether or not player i has been corrupted. If the adversary ever asks a **Corrupt** query and obtains player i's secret key as a result, δ_i is set to **corrupt**.

The list **LST** keeps track of *game related* local session states. While the list **SST** tells us about what's happening in each session as the *players* engage in the protocol, this list tells us about what's been corrupted in each session as a result of *adversarial* interference. In each session, the owner, the peer, or the session itself can all be corrupted. For each local session identifier $\ell = (i, j, k)$ its game related local session state consists of the following:

- $\boldsymbol{\delta}_{\mathbf{owner}} \in \{\mathbf{honest}, \mathbf{corrupt}\}$: tells us whether or not the owner was corrupted before the session terminated, i.e. while $\ell.\mathbf{accept}$ is set to $\perp$. Initialized to **honest**.

- $\boldsymbol{\delta}_{\mathbf{peer}} \in \{\mathbf{honest}, \mathbf{corrupt}\}$: tells us whether or not the peer was corrupted while $\ell.\mathbf{accept}$ is set to $\perp$. Initialized to **honest**.

- $\boldsymbol{\delta}_{\mathbf{sess}} \in \{\mathbf{fresh}, \mathbf{revealed}\}$: tells us whether or not the session key for this session was revealed to the adversary. Initialized to **fresh**.

As with the list **SST** of session ℓ's local session states, we will use the notation $\ell.\boldsymbol{\delta}_{\mathbf{owner}}$, etc, to refer to elements of its game related local session states.

The idea is that as the players run the protocol and generate new session keys, the adversary submits **Reveal** and **Corrupt** queries to the behavior algorithm to learn

players' private key and current and past session keys. To keep track of what, in each session, has been revealed (if anything) we update each session's game related session state.

We leave the definition of the list **MST** of global game related information to when we describe the key secrecy security game in Section 3.5.

To set up our security game, we run two procedures that populate the items on each list with their initial values:

1. $(\mathbf{SST}, \mathbf{EST}) \leftarrow \mathbf{setupE}(\mathbf{LSID}, \mathbf{kg}, 1^\lambda)$: This procedure uses **kg** and the security parameter 1^λ to provide the players with their public and private keys. It also initializes each component of the protocol related local session state for all sessions (i.e. for session ℓ it will set $\ell.\mathbf{accept}$ and $\ell.\mathbf{sid}$ to $\perp$); in other words, it populates the list **SST** for each session. Finally for each tuple $(i, \mathbf{pk}_i, \mathbf{sk}_i, \delta_i) \in \mathcal{L}_{\mathbf{keys}}$ it sets adds in each user's public and private key and sets δ_i to **honest**, populating the list **EST**.

2. $(\mathbf{LST}, \mathbf{MST}) \leftarrow \mathbf{setupG}(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, 1^\lambda)$: to set up game related components. This procedure initializes each component of the game related local session state for all sessions (i.e. for session ℓ it sets $\ell.\boldsymbol{\delta}_{\mathbf{owner}}$ to **honest**), populating the list **LST** for each session in **LSID** It also will initialize all components of the list **MST**, and we will describe in Section 3.5 what the initial values of this list will be.

3.3 Session Partnering

As we mentioned earlier in Section 3.1, two parties engaging in the key exchange protocol together each have their own session and their session IDs should be the same after terminating. We formalize this notion by defining *session partnering*. We use the definition from [BCP01]: two sessions ℓ and ℓ', $\ell \neq \ell'$, are partners if and only if $\ell.\mathbf{sid} = \ell'.\mathbf{sid}, \ell.\mathbf{sid} \neq \perp$ and $\ell'.\mathbf{sid} \neq \perp$. In other words, the two sessions are administratively different and have the same, non-trivial session ID.

3.4 Formal Game Definition

Now we can formally define what a security game is as specified in [dSGFW20], how the game is actually played, and how the adversary can win. Throughout we said the goal of the adversary was to trigger some "bad event"; this "bad event" is defined by a logical statement P that we evaluate on the game states **LSID**, **SST**, **EST**, **LST**, **MST**. Different security goals (and the games played to determine whether or not a protocol reaches those goals) have different logical statements that define their "bad event"; in Section 3.5 we will provide the logical statement that defines the key secrecy game's "bad event". Let b be the outcome of the logical statement: if $b = 1$ then the adversary has won and if $b = 0$ then the adversary has lost.

Let Q be the set of possible queries the adversary can ask. The behavior algorithm χ evaluates the queries the adversary sends using the **Valid** predicate, which will return **true** if the query is valid (and so the behavior algorithm will give the appropriate reply to the query) and **false** otherwise (and so the behavior algorithm

will not reply to the query). The **Valid** predicate can differ in formulation between security games and we will provide the exact formulation of this predicate for the key secrecy security game in the next section.

Now we can formally define a security game: a *security game* G maintains a state

$$(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, \mathbf{LST}, \mathbf{MST})$$

and is defined by the tuple $(\mathbf{setupE}, \mathbf{setupG}, Q, \mathbf{Valid}, \chi, P)$. The entire process, from giving everyone their public and private keys and initializing all values for all the session states to determining whether or not the adversary has won the game, is called an experiment. More precisely, an *experiment* is parameterized by a protocol $\boldsymbol{\pi}$, an adversary $\mathcal{A}$, and a game G, and is executed as follows:

1. We first set up the security game by running the two procedures mentioned earlier: $(\mathbf{SST}, \mathbf{EST}) \leftarrow \mathbf{setupE}(\mathbf{LSID}, \mathbf{kg}, 1^\lambda)$ to set up the protocol related components and $(\mathbf{LST}, \mathbf{MST}) \leftarrow \mathbf{setupG}(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, 1^\lambda)$ to set up the game related components.

2. The adversary submits queries which are processed by the behavior algorithm χ and the **Valid** predicate.

3. When $\mathcal{A}$ terminates, $b \leftarrow P(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, \mathbf{EST}, \mathbf{MST})$ is evaluated by the experiment and then outputs it.

As mentioned before, the various protocol and game related session states are updated accordingly in response to the queries the adversary sends. We will discuss what the adversary terminating looks like in the context of the key secrecy security game in the next section.

An experiment involving a security game G, an adversary $\mathcal{A}$, a protocol $\boldsymbol{\pi}$ and security parameter λ is denoted as

$$\mathrm{Exp}^{G}_{\mathcal{A},\pi}(1^{\lambda})$$

We denote the outcome of the experiment as $\mathrm{Exp}^{G}_{\mathcal{A},\pi}(1^{\lambda}) = b$, recalling that $b = 1$ means the adversary has won and $b = 0$ means the adversary has lost.

3.5 The Key Secrecy Security Game

Now we can define the key secrecy game in particular. To do so we will go over the list **MST** of global game related information, the **Test** query and an additional query called **Guess**, and the logical statement P that defines this game's "bad event" the adversary is trying to trigger.

In the key secrecy game, the adversary is trying to guess the session key for any past or current session. First she picks a session ℓ where $\ell.\boldsymbol{\delta}_{\mathbf{owner}} = \ell.\boldsymbol{\delta}_{\mathbf{peer}} = \mathbf{honest}$ and where $\ell.\boldsymbol{\delta}_{\mathbf{sess}} = \mathbf{fresh}$. From there, the behavior algorithm will either give her the actual session key $\ell.\mathbf{key}$ or a random element of $\mathcal{D}$, the set of all possible session keys (also known as the session key distribution) in $\mathbb{F}_q^n$, and she must determine whether or not she's been given the real session key. There's a 50% chance she's been given $\ell.\mathbf{key}$ and a 50% chance she's been given some random element. If she guesses correctly, she wins the game and has triggered the "bad event" because the key is no longer secret and the game ends; if she guesses incorrectly, she loses and the game ends. Since the adversary has a 50% chance of winning just by randomly guessing, we are interested in computing how much better than 50% the adversary can do by sending

out queries and harvesting the responses from those queries before she tests herself. In other words, we are interested in knowing how much worse off the players running the vector space projection protocol are if the adversary is able to get her hands on someone's private key or past session keys. We're concerned about how much better she can do than randomly guessing, about whether or not any past, present, or future session keys will stay secret.

We denote the key secrecy game by $G_{\mathrm{BRSec},\mathcal{D}}$. We use the protocol and game related local session states **SST** and **LST** and global protocol related information **EST** as described in Section 3.2. The list **MST** of global game related information contains the following elements: two bits b_{test}, initialized to 0 or 1 at random, and b_{guess}, initialized to $\perp$, and a session identifier $\ell_{\mathrm{test}} \in \mathbf{LSID}$ initialized to $\perp$. If $b_{\mathrm{test}} = 0$ the behavior algorithm will give the adversary a random element of $\mathcal{D}$ when she's testing herself; if $b_{\mathrm{test}} = 1$ then she gets the actual session key. The bit b_{guess} stores her guess, which she sends to the behavior algorithm when she's made a decision. Finally, if ℓ is the session she chooses to test herself with then $\ell_{\mathrm{test}} \leftarrow \ell$.

As mentioned at the beginning of this chapter, there's an additional query **Test** that is part of this security game. After the adversary has submitted as many of the other queries (**Send**, **Reveal**, **Corrupt**) as she likes in order to learn information about the protocol and the players, she sends a **Test** query that contains the session ℓ she wants to test herself with to the behavior algorithm. If ℓ is uncorrupted and the peer and owner are honest, $\ell_{\mathrm{test}} \leftarrow \ell$ and the algorithm will return to her ℓ.key or a random element of $\mathcal{D}$. She will send her guess back using the **Guess** query, which will set b_{guess} to be 0 (not the session key) or 1 (is the session key) accordingly.

There are additional requirements we add to ℓ if the adversary wants to test

herself with ℓ. As mentioned earlier in this section, we require that $\ell.\boldsymbol{\delta}_{\mathbf{owner}} = \ell.\boldsymbol{\delta}_{\mathbf{peer}} = \mathbf{honest}$ and $\ell.\boldsymbol{\delta}_{\mathbf{sess}} = \mathbf{fresh}$. We also require that if there is a session ℓ' that's partnered to ℓ (recall that two partner sessions have the same session ID) then $\ell'.\boldsymbol{\delta}_{\mathbf{sess}} = \mathbf{fresh}$ as well. In other words, both the session and its partner session are uncorrupted; otherwise the adversary could first send a **Reveal** query to ℓ' before sending a **Test** query containing ℓ and trivially win the game since both sessions would derive the same session key. Finally, in the event that $\ell'.\mathbf{id} \neq \ell.\mathbf{pid}$, we require that $\ell'.\mathbf{id}$ is still engaging in the protocol correctly; that is, they are following all the protocol steps in the correct order. (In the literature they would say that $\ell'.\mathbf{id}$ is *honest*, but for the sake of clarity we will only use honest to mean that their private key is still private). We add this condition in to account for the scenario in which a protocol cannot guarantee authentication and the owner of ℓ cannot be certain who their intended peer is. In this case, if their conversation partner is still following the protocol correctly, we would still want their session key to be secret, so we add in the previous requirement.

A session ℓ is *fresh* if and only if it has met all these conditions. We specify all these conditions because the **Valid** predicate for this game only requires that the adversary sends exactly one **Test** query to a session that's derived a key and one **Guess** query. In other words, the **Valid** predicate will only check that the adversary *can* ask the query, not whether or not by asking they give themselves a trivial win.

As we mentioned in Section 3.4, the last step in an experiment is to evaluate $b \leftarrow P(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, \mathbf{LST}, \mathbf{MST})$ and return b. The predicate **BRSec** we evaluate

for the key secrecy security game is the following:

$$b \leftarrow \mathbf{BRSec}(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, \mathbf{LST}, \mathbf{MST})$$

will evaluate to 0 if $b_{\text{guess}} \neq b_{\text{test}}$ or 1 if $b_{\text{guess}} = b_{\text{test}}$ if and only if $\mathbf{MST}.\ell_{\text{test}} \neq \perp$ and $\mathbf{MST}.\ell_{\text{test}}$ is fresh. In other words, our predicate is this: the experiment evaluates to 0 (adversary loses) or 1 (adversary wins) if and only if the adversary has chosen a session to test herself with and that session is fresh.

Finally we offer a precise definition of the adversary's advantage in the key secrecy game. For the key secrecy game G_{BRSec}, we have a protocol $\boldsymbol{\pi}$, a session key distribution $\mathcal{D}$, the set of all possible users $\mathcal{U}$ and participating users $\mathcal{I}$, and an adversary $\mathcal{A}$. Then the adversary's *advantage* is

$$\text{Adv}^{G_{\text{BRSec}},\mathcal{D}}_{\mathcal{A},\boldsymbol{\pi},\mathcal{U},\mathcal{I}} = 2\mathbb{P}\left(\text{Exp}^{G_{\text{BRSec}},\mathcal{D}}_{\mathcal{A},\boldsymbol{\pi},\mathcal{U},\mathcal{I}}(1^\lambda) = 1\right) - 1$$

The adversary's advantage is twice the probability she wins minus one, where the probability of her winning is the probability that the experiment evaluates to 1 (the logical statement $\mathbf{BRSec}(\mathbf{LSID}, \mathbf{SST}, \mathbf{EST}, \mathbf{LST}, \mathbf{MST})$ evaluates to 1). We note that when we are computing an adversary's advantage, we'll compute it in terms of the security parameter λ (for the vector space projection protocol, the security parameter is the size of the field q). In the next section we'll compute the advantage for four different adversaries, each using a different strategy to win the game, and show that each adversary's advantage is non-negligible as a function of the security parameter q.

Chapter 4

Strategies for the Adversary

As mentioned in the previous section, the adversary's advantage in the key secrecy game is

$$2\mathbb{P}\left(\mathrm{Exp}^{G_{\mathrm{BRSec}},\mathcal{D}}_{\mathcal{A},\pi,\mathcal{U},\mathcal{I}}(1^\lambda)=1\right)-1$$

and when computing her advantage, we want this advantage to be negligible as a function of the security parameter, which for the vector space projection protocol is q. As mentioned in the beginning of Chapter 3, a function $f(k)$ is negligible if for every $c>0$ there exists $k_c>0$ such that for all $k>k_c$, $f(k)<\frac{1}{k^c}$. In other words, $f(k)$ eventually decreases faster than any rational function (or its inverse increases faster than any polynomial function).

When we are trying to determine whether or not the key secrecy goal has been met, we need to determine whether or not *any* probabilistic polynomial time adversaries have a non-negligible advantage. If there's even one probabilistic polynomial-time algorithm that an adversary could use to learn the session key, then the protocol doesn't

provide key secrecy. Thus to show that a protocol provides key secrecy we'd need to show that all possible adversaries running all possible probabilistic polynomial-time algorithms have a negligible advantage. We will provide four probabilistic polynomial-time algorithms that allow for the adversary to win, which indicates that the vector space projection protocol does not provide key secrecy.

Here is the first such algorithm and we call it **Strategy 1**. This strategy guarantees victory, so her advantage is non-negligible as a function of q. Here is how it goes: the adversary sends **Reveal** queries to each session after each one terminates. She is able to keep track of which public vectors map to which session keys. Let $\vec{u}_i$ be the session key that was generated by projection $\vec{v}_i$ onto W. Then she knows that $\vec{u}_i \in W$, that $\vec{v}_i$ maps to $\vec{u}_i$ under the projection map, and that $\vec{v}_i - \vec{u}_i$ is in its kernel. She keeps sending **Reveal** queries until she has r linearly independent vectors in the image and $n - r$ linearly independent vectors in the kernel; now she can write down the matrix representation of the projection map and calculate all past session keys, the current session key, and all future session keys.

Strategy 1 reveals that it doesn't matter if r is private or public; if r is private then the adversary needs only to send a **Reveal** query to the sessions associated with n linearly independent public vectors, and then she can compute the matrix representation of the projection map. This strategy also tells us that it is imperative that at the very least the group manager must change W before r or $n-r$, whichever is larger, linearly independent vectors have been publicly released.

Next we have **Strategy 2**. This strategy also guarantees victory and so her advantage is again non-negligible as a function of q. The adversary employing this strategy will take advantage of the fact that if $\vec{u}_i$ is the projection of $\vec{v}_i$ onto W and

$\vec{v}_i$ is a linear combination of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{i-1}$, then $\vec{u}_i$ will be a linear combination of $\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_{i-1}$. To be more precise, if

$$\vec{v}_i = a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_{i-1}\vec{v}_{i-1},$$

where at least one coefficient is nonzero, then

$$\vec{u}_i = a_1\vec{u}_1 + a_2\vec{u}_2 + \cdots + a_{i-1}\vec{u}_{i-1}$$

Then the adversary needs only to wait for at most $n+1$ vectors to be publicly released; we are guaranteed to have a linear dependence amongst these $n+1$ vectors. Suppose that the ith vector is linearly dependent on the previous $i-1$ vectors for $i \leq n+1$, then she can determine the coefficients $a_1, a_2, \ldots, a_{i-1}$ for $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_{i-1}$ and send **Reveal** queries for the sessions that correspond to nonzero coefficients. From here, once she sends a **Test** query to the session corresponding to $\vec{v}_{i+1}$, she just has to check if

$$\vec{b} = a_1\vec{u}_1 + a_2\vec{u}_2 + \cdots + a_i\vec{u}_i.$$

What this strategy reveals to us is that if the group manager picks each vector $\vec{v}_i$ to be linearly independent of the first $i-1$ vectors, once $n+1$ session keys have been generated, if enough past session keys are leaked then an adversary can compute all past session keys, the current session key, and all future session keys *without knowing W or the projection map.* If the group manager isn't deliberately ensuring linear independence, then it's possible that the adversary can compute the current session

key if $\vec{v}_i$ for $i < n + 1$ is linearly dependent on the first $i - 1$ vectors, again without knowing W or the projection map.

What **Strategy 1** tells us is that the adversary can figure out the projection map, and thus compute all session keys, just by matching enough past public vectors with past session keys, but what **Strategy 2** tells us is that she doesn't need to go that far: she could simply use past session keys and the linear dependence between them to figure out the current session key and all past and future ones as well if she waits long enough. These strategies indicate that this protocol is very sensitive to data leaks, and **Strategy 2** in particular shows that this would remain the case even if we place the projection map with any other linear map.

Third is **Strategy 3** which also guarantees victory and thus also gives the adversary a non-negligible advantage. Here is how this strategy goes: first she sends a **Corrupt** query to any user, which gives her that user's private key. Then she picks a fresh session ℓ and sends a **Test** query (for example, she could pick any session that doesn't involve the corrupted user). Since she now has one user's private key, she can compute all past, present, and future session keys and will be able to determine ℓ.**key**, ℓ's session key.

This strategy tells us that only one person's private key leaking is enough to compromise key secrecy for the entire group, as the group manager is now forced to reissue private keys for *all* users in the entire group. While just two or three past session keys leaking isn't enough for an adversary to piece together the projection map, and it's unlikely that linear dependence between session keys will show up when only two or three session keys have been leaked, it's troubling to note that only one leaked private key will compromise key secrecy for everyone.

Finally we have **Strategy 4**. Let $\vec{b}$ be the vector that the adversary receives when she sends a **Test** query to a fresh session. Let $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m$ be the vectors the group manager publicly released to generate past session keys. If $\vec{b}$ is a multiple of any of these publicly released vectors, then she'll know immediately that $\vec{b}$ is not the session key and therefore can send a **Test** query and guess "no". None of the publicly released vectors are in the secret subspace W; if a session key were to be a multiple of one of the publicly released vectors, then that publicly released vector would also be in W. If $\vec{b}$ is not a multiple of any of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m$, she guesses "yes" and "no" with equal probability.

Now we'll compute the probability that an adversary employing this strategy wins, which will let us compute her advantage. For simplicity from now on, we abbreviate the condition that "$\vec{b}$ is a multiple of one of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m$" by "$\vec{b}$ is a multiple" and analogously for the condition that $\vec{b}$ is not a multiple of any of them.

$$\begin{aligned}\mathbb{P}(\text{win}) =& \mathbb{P}(\text{win} \mid \vec{b} \text{ is a multiple})\mathbb{P}(\vec{b} \text{ is a multiple}) + \\ & \mathbb{P}(\text{win} \mid \vec{b} \text{ isn't a multiple})\mathbb{P}(\vec{b} \text{ isn't a multiple})\end{aligned}$$

The probability that she wins given that $\vec{b}$ is a multiple is 1, since she'll know right away that $\vec{b}$ is not the session key. The probability that she wins if $\vec{b}$ is not a multiple is 0.5, so we just need to compute the probability that $\vec{b}$ is a multiple and the probability that it isn't.

Here we compute the probability that $\vec{b}$ is a multiple of one of the publicly released vectors. We assume that none of the publicly released vectors are multiples of each

other; given that n and q will both be large, this is a reasonable assumption to make.

$$\begin{aligned}
\mathbb{P}(\vec{b} \text{ is a multiple}) &= \sum_{i=1}^{m} \mathbb{P}(\vec{b} \text{ is a multiple of } \vec{v}_i) \\
&= \sum_{i=1}^{m} \mathbb{P}(\vec{b} \neq \text{ session key and is a multiple of } \vec{v}_i) \\
&= \sum_{i=1}^{m} \mathbb{P}(\vec{b} \neq \text{ session key}) \mathbb{P}(\vec{b} \text{ is a multiple of } \vec{v}_i \mid \vec{b} \neq \text{ session key}) \\
&= \sum_{i=1}^{m} \frac{1}{2} \mathbb{P}(\vec{b} \text{ is a multiple of } \vec{v}_i \mid \vec{b} \neq \text{ session key}).
\end{aligned}$$

The probability that $\vec{b}$ is a multiple of $\vec{v}_i$ given that $\vec{b}$ is not the session key is given by:

$$\frac{q-2}{q^n - 2 - m}.$$

We assume here that $\vec{b}$ is chosen uniformly at random from $\mathbb{F}_q^n$, avoiding the zero vector, the actual session key, and all the vectors $\vec{v}_1, \ldots, \vec{v}_m$, which leaves us with $q^n - 2 - m$ choices total for $\vec{b}$. Of those choices, $q-2$ of them will be multiples of $\vec{v}_i$: there are q multiples of $\vec{v}_i$ total, one of which is the zero vector and one of which is $\vec{v}_i$ itself.

Thus our probability that $\vec{b}$ is a multiple of one of the m publicly released vectors is:

$$\sum_{i=1}^{m} \frac{1}{2} \frac{q-2}{q^n - 2 - m}.$$

We can compute the probability that $\vec{b}$ is not a multiple of one of the m publicly

released vectors by subtracting from 1:

$$\mathbb{P}(\vec{b} \text{ is not a multiple}) = 1 - \sum_{i=1}^{m} \frac{1}{2} \frac{q-2}{q^n - 2 - m}.$$

Now we can compute the probability the adversary wins:

$$\begin{aligned}
\mathbb{P}(\text{win}) =& \mathbb{P}(\text{win} \mid \vec{b} \text{ is a multiple}) \mathbb{P}(\vec{b} \text{ is a multiple}) + \\
& \mathbb{P}(\text{win} \mid \vec{b} \text{ isn't a multiple}) \mathbb{P}(\vec{b} \text{ isn't a multiple}) \\
&= \sum_{i=1}^{m} \frac{1}{2} \frac{q-2}{q^n - 2 - m} + \frac{1}{2} \left(1 - \sum_{i=1}^{m} \frac{1}{2} \frac{q-2}{q^n - 2 - m} \right) \\
&= \frac{1}{2} + \frac{1}{2} \left(\sum_{i=1}^{m} \frac{1}{2} \frac{q-2}{q^n - 2 - m} \right) \\
&= \frac{1}{2} + \frac{1}{2} \left(\frac{m(q-2)}{2(q^n - 2 - m)} \right).
\end{aligned}$$

Then her advantage is:

$$2 \left(\frac{1}{2} + \frac{1}{2} \left(\frac{m(q-2)}{2(q^n - 2 - m)} \right) \right) - 1 = 1 + \frac{m(q-2)}{2(q^n - 2 - m)} - 1 = \frac{m(q-2)}{2(q^n - 2 - m)}.$$

Her advantage is non-negligible as a function of q but negligible as a function of n; thus, in order for an adversary employing this strategy to have a non-negligible advantage we need to change the security parameter to n, the dimension of the vector space, instead of q. However, as we've seen with the previous strategies an adversary can still win regardless of what n is when there are enough data leaks.

Thus, because each of the above strategies gives an adversary a non-negligible advantage, the vector space projection protocol does not provide key secrecy and we do not recommend its implementation as a group key exchange protocol.

Chapter 5
Bibliography

[BCP01] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange — the dynamic case. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 290–309, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BCP02a] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings*, pages 321–336. Springer, 2002.

[BCP02b] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attacks. In *Advances in Cryptology—ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings 8*, pages 497–514. Springer, 2002.

[BCPQ02] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. *Proceedings of the ACM Conference on Computer and Communications Security*, July 2002.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 51–62, 2011.

[BGVS07] Jens-Matthias Bohli, María Isabel González Vasco, and Rainer Steinwandt. Secure group key establishment revisited. *International Journal of Information Security*, 6:243–254, 2007.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, pages 232–249, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[Che18] Changhao Chen. Projections in vector spaces over finite fields. In *Annales Academiæ Scientiarum Fennicæ Mathematica*, volume 43, pages 171–185, 2018.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*, pages 453–474. Springer, 2001.

[COG21] Baris Celiktas, Enver Ozdemir, and Sueda Guzey. An inner product space-based hierarchical key assignment scheme for access control. `https://www.techrxiv.org/articles/preprint/An_Inner_Product_Space-Based_Hierarchical_Key_Assignment_Scheme_for_Access_Control/16577402`, 2021.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. In *IEEE Transactions on Information Theory*, volume 22, 1976.

[dSGFW20] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 288–303, 2020.

[FGSW16] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 452–469, 2016.

[GBNM08] M. Choudary Gorantla, Colin Boyd, Juan Manuel González Nieto, and Mark Manulis. Modeling key compromise impersonation attacks on group key exchange protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(4):1–24, 2008.

[GKO21] Sueda Guzey, Gunes Karabulut Kurt, and Enver Ozdemir. A group key establishment scheme. `https://arxiv.org/abs/2109.15037`, 2021.

[Har13] Lein Harn. Group authentication. *IEEE Transactions on Computers*, 62(9):1893–1898, 2013.

[ITW82] I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, 1982.

[KLL04] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee. Constant-round authenticated group key exchange for dynamic groups. In *Advances in Cryptology-ASIACRYPT 2004: 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004. Proceedings 10*, pages 245–259. Springer, 2004.

[KS05] Jonathan Katz and Ji Sun Shin. Modeling insider attacks on group key-exchange protocols. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 180–189, 2005.

[PRSS21] Bertram Poettering, Paul Rösler, Jörg Schwenk, and Douglas Stebila. SoK: Game-based security models for group key exchange. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, pages 148–176, Cham, 2021. Springer International Publishing.