E. Jongsma

Algebraic Manipulation Detection Codes

Bachelorscriptie, 6 maart 2008

Scriptiebegeleider: prof.dr. R.J.F. Cramer

Mathematisch Instituut, Universiteit Leiden

Contents

1 Introduction

This thesis is about a subject in the field of cryptology called algebraic manipulation detection codes, or AMD-codes for short. We will discuss different ways of constructing these codes, and prove some theorems about them. Parts of this thesis will consist of theorems and definitions which are already known. These are mostly taken from the paper by R. Cramer, S. Fehr and C. Padró [1]. We will also introduce several concepts from other fields of cryptology and mathematics. For example, we will discuss authentication codes and some theorems known from combinatorics. However, the aim is to give a more elaborate explanation of the known constructions, and show some new results. But before we will talk about exactly what these codes are and what they do, we will discuss cryptology in general.

There are a couple of things to keep in mind when talking about this field. First, one must always take into account that cryptology has two goals. One goal is to make certain things secure against a so-called adversary. Of course, there are many different kinds of security, but we will not talk about those now. Think for example about secure storage, secure message transmission, etc. The other goal is to break this security, usually called cryptanalysis. Therefore, someone working on cryptology will have to switch all the time between two ways of thinking. He will try to create something that has a certain functionality (with security), and then switch to the point of view of the adversary and try to break the security he has just created. Only when the last step fails will he have something useful. This is something to keep in mind, especially when looking at error probabilities. When we try to determine what chance our adversary has of breaking our security, we will look at it from his point of view, which might sometimes be confusing.

For example, look at one-time pad encryption. This is a very simple idea: Alice wants to send a secret message to Bob. However, there is a chance that this message will be intercepted. In order to make sure that Eve, who is listening on the channel between Alice and Bob, cannot find out what the message is, Alice encrypts it. We assume here that Alice and Bob have a secret key which only they two know about. Then, from the point of view of Alice and Bob, it does not matter if the message is encrypted or not, since they can read it anyway (by decoding if necessary). However, from the point of view of Eve, the encoded message is just that (since she does not have the key used to encode it), and she will have to find another way to break the encryption.

In this case, the difference between the points of view of both is very clear. In more involved situations however, it sometimes becomes difficult to keep track of what our adversary does or does not know. In this thesis, you will encounter some of these situations. Hopefully, the explanation given will be enough to get a good feeling of what is happening.

2 What are AMD codes?

Before giving an exact definition, we will first look at the general idea behind an AMD-code. First, we must consider the notion of an *abstract storage device*, called $\Sigma(G)$. This storage device holds an element g from an finite abelian group G, in such a way that our adversary does not have any clue about the stored information. More formally, we could say that the a priori information our adversary has about the stored information does not increase when he is allowed access to $\Sigma(G)$. However, he *is* allowed to add another element $\delta \in G$ to the one already stored, so that $g+\delta$ is now the element in our device. Note that the choice of δ our adversary must make can only be based on what he already knew about g before it was stored in the device. This idea might seem a little strange at first, but later it will become clear why we use this $\Sigma(G)$ and how it models certain scenarios we encounter in practice. Now suppose our adversary has indeed added a $\delta \neq 0$ to our $g \in \Sigma(G)$. Then, if we open the device, $g+\delta$ comes out and we have no way of finding out what δ or g is. So, although our adversary has no information about the data stored in the device, he can make sure that when we open it, we retrieve a value different from the one we stored. It is clear this is an unwanted situation. Therefore, we need to find a way to store data in $\Sigma(G)$ in such a way that when our adversary alters the data, we can detect it.

This is where AMD codes come into play. What an AMD-code does is probabilistically encode a source s from a predetermined set $\mathcal{S}$ as an element $g \in G$. Given such an encoding g, there is a unique s which encodes to it, i.e. g is uniquely decodable. Instead of just putting the information s we want to store in our device, we now store the encoding of our information, g. If g is altered by our adversary, we will (by the properties of the AMD-code) detect this except with a small error probability ε. Note that there is no need for a secret key, i.e., the information stored in $\Sigma(G)$ alone is enough to recover the original data if no alteration by an adversary is detected.

There are two types of AMD codes, called weakly and strongly secure. In the weakly secure case, the source s is uniformly distributed from the point of view of the adversary. In the strongly secure case, sometimes just called secure for simplicity, security still holds if our adversary knows s. In fact, he can even choose it himself to optimise his chances of fooling us. Note that in both cases, the information stored in $\Sigma(G)$ is hidden from the adversary, the only difference lies in the fact whether or not they know the source state. In order to get this kind of security, the size of the encoding space G must be bigger then that of the source space S. In 2.2, a lower bound will be shown on this increase in size. As we will see, the increase of size is not always equal to the lower bound. Therefore, we introduce the notion of overhead and effective overhead. These determine the increase of the storage space needed to store the encoding instead of the source. They also allow us to compute how far we are from the lower bound on the size of the encoding space.

2.1 Definition of an AMD-code

Let $m, n \in \mathbb{N}^+$, with $m \leq n$, and let $0 < \varepsilon \leq 1$.

Definition 2.1.1. *An (m, n)-algebraic manipulation detection code, or (m, n)-AMD code for short, is a probabilistic encoding function $E : \mathcal{S} \to G$ from a set $\mathcal{S}$ of size m into a finite abelian group G of order n, together with a decoding function $D : G \to \mathcal{S} \cup \{\perp\}$ such that $D(g) = s$ if $g = E(s)$ for some $s \in S$, and $\perp$ otherwise.*

The idea behind our decoding function D is that if we decode an encoding of an element $s \in \mathcal{S}$, we always get s, since we want unique decodeability. However, if we try to decode an element $g \in G$ that is not an encoding, we know that our stored data has been corrupted and we output the "T upside down", known as "bot". Note that if we want our adversary's chances of fooling us to be small, the probability that $E(s) + \delta = E(s')$ must be small. In words: if the adversary adds an element δ to our encoding, the chance that the result is an encoding of a different source state s' must be small. This idea is written down more precisely in the following two definitions:

Definition 2.1.2. *An (m,n)-AMD code is called weakly ε-secure if the following holds: For s sampled at random from $\mathcal{S}$, and for δ sampled from G according to some distribution independent of s and $E(s)$, the probability that $D(E(s) + \delta) = s' \neq s$ is at most ε.*

Note that in this definition, the source state s is uniformly distributed from the point of view of our adversary. Therefore, his distribution on δ is independent of not only $E(s)$, but also of s. The case where our adversary is allowed to know s, and is in fact even allowed to pick one to optimise his probability of breaking the code, is defined below:

Definition 2.1.3. *An (m,n)-AMD code is called strongly ε-secure, or simply ε-secure, if the following holds: For any $s \in \mathcal{S}$, and for δ sampled from G according to some distribution independent of $E(s)$ when given s, the probability that $D(E(s) + \delta) = s' \neq s$ is at most ε.*

Finally, an AMD code is called *systematic* if the source set $\mathcal{S}$ is a group and the encoding is of the form

$$\begin{aligned} E : \mathcal{S} &\to \mathcal{S} \times \mathcal{G}_1 \times \mathcal{G}_2 \\ s &\mapsto (s, x, f(x, s)), \end{aligned}$$

for a function f, and where $x \in_R \mathcal{G}_1$.

At this point, the reader very much interested in the motivation behind using AMD codes could skip ahead to chapter 3. This chapter will continue with some theorems about AMD codes and an example.

2.2 Lower bound on n

It is obvious from the definition that n must be at least as large as m (else we would not have unique decodeability). The question however is: how much bigger does n have to be? In this section, lower bounds on n will be proved for the weakly and strongly secure case.

Let $m, n \in \mathbb{N}^+$, with $m \leq n$, and let $0 < \varepsilon \leq 1$. Also, let (E, D) be the encoding respectively decoding function for an (m, n)-AMD code, with $E : \mathcal{S} \to G$ and $D : G \to \mathcal{S} \cup \{\perp\}$. We will also need $D^{-1}(s)$, which is defined as the set $\{g \in G : D(g) = s\}$. It is clear that for all $s, s' \neq s \in \mathcal{S}$ the following holds: $D^{-1}(s) \cap D^{-1}(s') = \emptyset$ and $|D^{-1}(s)| \geq 1$.

We are now ready to state and prove the lower bounds:

Theorem 2.2.1. *Any weakly ε-secure (m,n)-AMD code satisfies:*

$$n \geq \frac{m-1}{\varepsilon} + 1$$

Proof. In the weakly secure case, s is uniformly distributed. We sample $\delta \neq 0$ at random from G, independent of s and $E(s)$. From definition 2.1.2 we know that the probability that $D(E(s) + \delta) = s' \neq s$ is at most ε. Writing $\boldsymbol{E}$ for the expectation over s and $E(s)$, it follows that

$$\varepsilon \geq \frac{\boldsymbol{E}[|\delta \in G : E(s) + \delta \in \cup_{s' \neq s} D^{-1}(s')|]}{n-1} = \frac{\boldsymbol{E}[\sum_{s' \neq s} |D^{-1}(s')|]}{n-1} \geq \frac{m-1}{n-1}$$

and then we know $n \geq \frac{m-1}{\varepsilon} + 1$. □

The proof for the strongly secure case is quite similar:

Theorem 2.2.2. *Any (strongly) ε-secure (m,n)-AMD code satisfies:*

$$n \geq \frac{m-1}{\varepsilon^2} + 1$$

Proof. In the strongly secure case, s is fixed. Then, we know that $|D^{-1}(s)| \geq \frac{1}{\varepsilon}$. This is not hard to see, for example, take $\varepsilon = \frac{1}{3}$. Then $|D^{-1}(s)| \geq 3$ has to be true, because if $|D^{-1}(s)| = 2$, $E(s)$ is known with probability $\frac{1}{2}$ (since s is fixed) and therefore the code is broken. Now, we arrive at an expression similar to the one above:

$$\varepsilon \geq \frac{\boldsymbol{E}[|\delta \in G : E(s) + \delta \in \cup_{s' \neq s} D^{-1}(s')|]}{n-1} = \frac{\boldsymbol{E}[\sum_{s' \neq s} |D^{-1}(s')|]}{n-1} \geq \frac{m-1}{\varepsilon(n-1)}$$

and then we know $n \geq \frac{m-1}{\varepsilon^2} + 1$. □

Note that in this case, $\boldsymbol{E}$ is the expectation over $E(s)$ when given s, since our adversary knows s.

Armed with these bounds, we are now ready to look at the overhead and effective overhead.

2.3 Overhead and effective overhead

From the previous section, it is clear that an encoding of a source state is "bigger" than the source state itself. The difference in size between the two is a measure of how efficient our code is, which we will call the overhead.

Definition 2.3.1. *The overhead $\bar{\omega}$ of an (m,n)-AMD code is defined as $\bar{\omega} = \log(n) - \log(m)$.*

The log here is actually the base 2 log, so that $\log(n)$ gives us the amount of bits needed to store n in a computer (using binary numbers). The problem here is that the notion of overhead is not very useful in practice. Usually, one wants to specify the error probability and bit length of the source state that is to be encoded, and then look at how much extra storage space is needed to achieve this. To do this, we need a different way of looking at AMD codes. Instead of looking at one specific instance of an AMD code, we now look at a whole class of codes, each one tailored to fit parameters concerning input and error probability. This gives us the following definition:

Definition 2.3.2. *An AMD code family is a class of AMD codes such that for any $\kappa, \ell \in \mathbb{R}^+$ there exists a (m,n)-AMD code in that class with $m \geq 2^\ell$ and $\varepsilon \leq 2^{-\kappa}$.*

Now, we can look at the effectiveness of codes with specified error probability and bit length of the source state in the following way:

Definition 2.3.3. *The effective overhead $\bar{\omega}^*$ of an AMD code family is a function that maps $\kappa, \ell \in \mathbb{R}^+$ to $\bar{\omega}^*(\kappa, \ell) = \min(\log(n) - \log(m))$, where the minimum is taken over all (m,n)-AMD code which satisfy $m \geq 2^\ell$ and $\varepsilon \leq 2^{-\kappa}$.*

We will from now on just talk about the effective overhead of an AMD code, meaning that we actually look at the effective overhead of the AMD code family obtained by varying the parameters of the code.

Since we know lower bounds for n with respect to m (see previous section), we can also compute lower bounds for the effective overhead of AMD codes. Our goal will of course then be to find AMD codes that are as close as possible to these lower bounds.

Theorem 2.3.4. *The effective overhead of a weakly ε-secure AMD code is lower bounded by*

$$\bar{\omega}^*(\kappa, \ell) \geq \kappa - 2^{-\ell+1}$$

Proof. We have already shown that $n \geq \frac{m-1}{\varepsilon} + 1$ for any weakly secure (m,n)-AMD code (Theorem 2.3.1). Then, we know that $\log(n) \geq \log(\frac{m-1}{\varepsilon} + 1)$. Substituting this in our equation gives

$$\log(\frac{m-1}{\varepsilon} + 1) - \log(m) \geq \log(\frac{m-1}{\varepsilon}) - \log(m) = \log(\frac{m-1}{m}) - \log(\varepsilon)$$

Now, we know that $\varepsilon \leq 2^{-\kappa}$, so $-\log(\varepsilon) \geq -\log(2^{-\kappa})$ and thus

$$\log(n) - \log(m) \geq \kappa + \log(\frac{m-1}{m}) = \kappa + \log(1 - \frac{1}{m}).$$

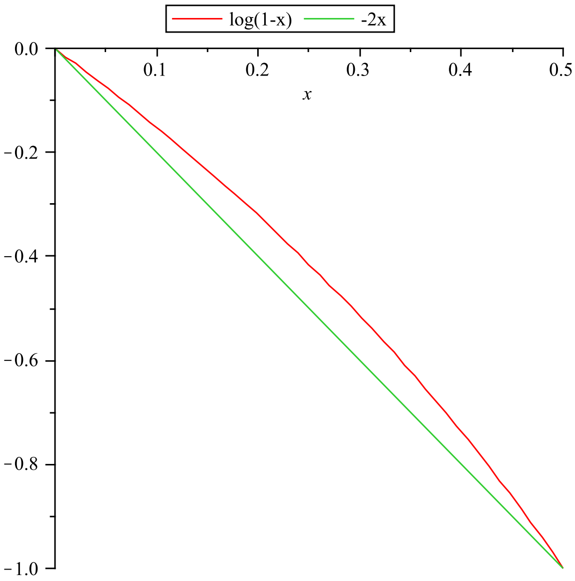
Since $m \geq 2^{-\ell}$, we know $-\frac{1}{m} \geq -2^{-\ell}$ and $\log(n) - \log(m) \geq \kappa + \log(1 - 2^{-\ell})$. We now need the following lemma:

Lemma 2.3.5. *For $0 \leq x \leq \frac{1}{2}$, the following inequality holds:*

$$\log(1-x) \geq -2x$$

Proof. First, note that equality holds for $x = 0$ and $x = \frac{1}{2}$. We know that $-2x$ has a constant first derivative which is -2. Now, the first derivative of $\log(1-x)$ is $\frac{-1}{(1-x)\ln(2)}$. Since $0 \leq x \leq \frac{1}{2}$, the first derivative of the log is negative on the interval and therefore the original function is decreasing. The second derivative then becomes $\frac{-1}{(1-x)^2\ln(2)}$, which is clearly always negative on our interval. This means $\log(1-x)$ is decreasing at an increasing rate between $x = 0$ and $x = \frac{1}{2}$, and therefore must be above $-2x$ between the bounds of the interval. □

See below for a plot of the two functions:



We will now use this lemma to simplify our previously found expression:

$$\log(n) - \log(m) \geq \kappa + \log(1 - 2^{-\ell}) \geq \kappa - 2 \cdot 2^{-\ell} \geq \kappa - 2^{-\ell+1}$$

□

The bound and its proof is much the same for strongly secure codes:

Theorem 2.3.6. *The effective overhead of a strongly ε-secure AMD code is lower bounded by*

$$\bar{\omega}^*(\kappa, \ell) \geq 2\kappa - 2^{-\ell+1}$$

Proof. We have already shown that $n \geq \frac{m-1}{\varepsilon^2} + 1$ for any strongly secure (m, n)-AMD code (Theorem 2.3.2). Then, we know that $\log(n) \geq \log(\frac{m-1}{\varepsilon^2} + 1)$. Substituting this in our equation gives

$$\log(\frac{m-1}{\varepsilon^2}+1)-\log(m) = \log(\frac{m-1+\varepsilon^2}{\varepsilon^2})-\log(m) = \log(m-1+\varepsilon^2)-\log(\varepsilon^2)-\log(m)$$

Now, we know that $\varepsilon \leq 2^{-\kappa}$, so $-\log(\varepsilon^2) \geq -\log(2^{-2\kappa})$ and thus

$$\log(n) - \log(m) \geq 2\kappa + \log(m - 1 + \varepsilon^2) - \log(m) = 2\kappa + \log(1 - \frac{1}{m} + \frac{\varepsilon^2}{m}).$$

Since $\frac{\varepsilon^2}{m}$ is very small, we will just ignore it, and the rest of the proof becomes analogous to that of the weakly secure case. Thus, our equation becomes:

$$\log(n) - \log(m) \geq 2\kappa - 2 \cdot 2^{-\ell} \geq 2\kappa - 2^{-\ell+1}$$

□

Since we know that $2^{-\ell+1}$ will usually be very small in practice, we can see that the minimal effective overhead of weakly resp. strongly secure AMD codes is essentially κ resp. 2κ. Our constructions will of course try to get as close as possible to this bound.

2.4 Example of a weakly secure AMD code

We will now present a simple example of an AMD code and its parameters:

Let $\mathbb{F}$ be a finite field of characteristic $q \neq 2$. Consider the following AMD-code, due to Cabello, Padró, and Sáez [6]:

$$\begin{aligned} E : \mathbb{F} &\rightarrow \mathbb{F} \times \mathbb{F} \\ s &\mapsto (s, s^2) \end{aligned}$$

Theorem 2.4.1. *The above (q, q^2)-AMD code E is weakly ε-secure, where $\varepsilon = \frac{1}{q}$. Its effective overhead is $\bar{\omega}^*(\kappa, \ell) \geq \max(\kappa, \ell)$.*

Proof. It is clear that E is an (q, q^2)-AMD code. Now, we need to determine ε. Fix an arbitrary translation $(\Delta s, \Delta s^2) \in \mathbb{F} \times \mathbb{F}$ with $\Delta s \neq 0$. Now, we need to count the number of different s that solve

$$\begin{aligned} (s + \Delta s)^2 &= s^2 + \Delta s^2 \\ s^2 + 2s\Delta s + (\Delta s)^2 &= s^2 + \Delta s^2 \\ 2s\Delta s &= \Delta s^2 - (\Delta s)^2 \\ s &= \frac{\Delta s^2 - (\Delta s)^2}{2\Delta s} \end{aligned}$$

There is clearly only one solution for this equation, since Δs^2 and $(\Delta s)^2$ are fixed. Therefore, our error probability ε becomes $\frac{1}{q}$.
Regarding the overhead, we see that $\bar{\omega} = \log(n) - \log(m) = \log(q^2) - \log(q) = \log(q)$. However, if we consider the effective overhead, we get parameters (κ, ℓ) and we need to find the code which satisfies $m \geq 2^{\ell}$ and $\varepsilon \leq 2^{-\kappa}$ with the lowest possible overhead. However, from the 2 given inequalities we get $q \geq 2^{\ell} \Rightarrow \log(q) \geq \ell$ and $q \leq 2^{-\kappa} \Rightarrow \log(q) \geq \kappa$. So, our effective overhead $\bar{\omega}^*$ is at least κ or ℓ, depending on which is bigger. □

In practice, our ℓ will usually be much larger than κ. Therefore, we can see that although the overhead seems close to optimal, the effective overhead can be very high in certain situations. Note that the decoding function is not listed above. This is done because it should be clear from the encoding function how the decoding function should work, i.e. we decode to s if the pair (s, s^2) is correct, and output $\perp$ otherwise. In the remainder of this thesis, the decoding function will only be listed if necessary.

3 Motivation for using AMD codes

A question we want to answer is of course: what are AMD codes good for? In the introduction of AMD codes we spoke about the ”abstract storage device”. We did not go into further detail about it then, but now we will see this device is not as abstract as it seems. Giving two practical applications, we will show that the properties of the abstract storage device are what they are for a purpose.

3.1 Using AMD codes for robust secret sharing

Secret sharing is a way to distribute a secret over multiple players, in such a way that certain subsets of them can reconstruct the original secret. For example, a (t, n)-threshold sharing scheme $(1 \leq t < n)$ will allow any group consisting of $p > t$ players to reconstruct the secret. If there are equal or less than t players present, no information at all is known about the secret. Note that t is considered to be a bound on the amount of players an adversary can corrupt, so that the adversary can never find out anything about the secret. There are many ways of doing this, but we will just consider a well known scheme known as Shamir's secret sharing scheme [5]. The below facts also hold for general linear secret sharing schemes. Let $\mathbb{F}$ be a finite field and let n be a positive integer with $|\mathbb{F}| > n$. Let $x_1, \ldots x_n \in \mathbb{F}$ be distinct, fixed, non-zero elements. Assume that the secret s is also an element of $\mathbb{F}$. Then the scheme works as follows:

- Select a polynomial $f(x) \in \mathbb{F}[X]$ uniformly random, with the property that $\deg f \leq t$ and $f(0) = s$.
- For $i = 1 \ldots n$, define
$$s_i = f(x_i),$$
the i-th share of the secret s.

Then each player i receives his or her share s_i. What we want from the scheme are the following two properties:

- Correctness: any subset of $p > t$ players should be able to reconstruct the secret s uniquely.
- Secrecy: any subset of $p \leq t$ players has no information at all about s.

What we mean by secrecy is that the knowledge about the secret s is not increased by putting the shares of $p \leq t$ players together. We will argue that Shamir's scheme has these two properties by using the Lagrange Interpolation Theorem:

Theorem 3.1.1. ***(Lagrange)*** *Let $\mathbb{F}$ be a field, and m a positive integer. Consider m points $(x, y) \in \mathbb{F}^2$ with distinct x-coordinates. Then there exists a unique polynomial $f(x)$ with $\deg f < m$ that passes through the given points, i.e. $f(x) = y$ for all given pairs (x, y).*

Since this is a well-known theorem that can be proved in many ways, we will omit the proof here. Instead, we will use it to show that our scheme has the desired properties.

- Correctness: any subset of $p > t$ players should be able to reconstruct the secret s uniquely. We know that our players each have a point on a polynomial with distinct x-coordinates. If we take a set of $t+1$ players, we can interpolate a polynomial $g(x)$ with $\deg g \leq t$ which passes through these points. We also know that $\deg f \leq t$, and f also passes through the points used to interpolate g. Then, by the uniqueness of the interpolating polynomial, we know $f = g$, and then $g(0) = f(0) = s$.

- Secrecy: any subset of $p \leq t$ players has no information at all about s. Consider a set of t players. Fix a guess $s' \in \mathbb{F}$ for the real secret s. We add the point $(0, s')$ to the points already known. Then, by the Lagrange Interpolation Theorem, there exists a unique polynomial $f_{s'}(x)$ with $\deg f \leq t$, which passes through all points our set of t players has. Therefore, from the point of view of the adversary, every guess for s is equally likely to be correct, and hence he gains no extra information about it.

It is immediately clear from the properties of the scheme that if our adversary controls $p > t$ players, he knows what the secret is. However, a possibility we must also consider is the fact that our adversary might try to trick us into reconstructing the wrong secret. This is where AMD codes come into play. Instead of sharing our secret s, we instead share an encoding $E(s)$. Then, if every player used in the reconstruction is honest, $E(s)$ will be produced and after decoding we get our original secret s. Now assume at least one player is corrupted, and sends in a different share at reconstruction time. Then, instead of reconstructing $E(s)$, we reconstruct $E(s) + \delta$. Note that the linearity of the scheme allows our adversary to choose this δ.

From this observation, it immediately follows that our adversary should not be rushing. A rushing adversary is allowed to wait until the honest players have submitted their shares (during reconstruction), before sending his own corrupt share(s). So, if our adversary is rushing, he can adopt the following strategy: first, he corrupts one player. He then waits for all the other players to submit their share in the reconstruction phase. He now has enough information to reconstruct $E(s)$. But if the adversary knows $E(s)$, and he can choose δ as described above, he can break the AMD code (and therefore the secret sharing scheme). In fact, he can even choose the s' that will be decoded. To do this, he first chooses an s'. He then computes $E(s')$ and $\delta = E(s') - E(s)$. Then, $E(s)+\delta$ is reconstructed to be $E(s')$, and s' will be decoded. Therefore, we must assume our adversary is non-rushing, i.e. he has to choose his corrupt shares before reconstruction begins. Then, by the properties of the AMD code, the cheating is detected except with a small error probability. This kind of secret sharing scheme is called *robust* [8, 9].

Unfortunately, our decoding function outputs $\perp$ if this happens, so we seem to have no way of recovering the original s. In secret sharing schemes, we know that we need $t < \frac{n}{3}$ in order to have perfect reconstruction of a secret. However, in a robust setting, $t < \frac{n}{2}$ is enough to reconstruct except with a small probability related to the error probability of the AMD code. This is easily seen: if $t < \frac{n}{2}$, then there is at least one set of $t+1$ players of which none is corrupt. Therefore, a (computationally rather inefficient) method of reconstruction is to just execute the reconstruction phase on each subset of qualified players [7]. Note that if our adversary succeeds in breaking the AMD code, we have a qualified set of players (at least one of which is corrupt), which reconstruct without detection of an error. This means the error probability of reconstruction is related to that of the AMD code used (and thus, the error probability of the robust scheme).

3.2 Using AMD codes for secure message transmission

A problem in cryptology which is related to that of secret sharing is secure message transmission [10, 11]. In this setting, we have a sender S and a receiver R. S and R are connected by a synchronous network of n noiseless channels. Of course, there is also an adversary trying to corrupt these channels. If a channel is corrupted by an adversary, it means he has total control over it. He can change messages that are sent over it, inject messages of his own and schedule delivery of the messages any way he likes. We assume that channels that are not corrupted are private, i.e., the adversary has no knowledge at all about what is transmitted over them. There is an upper bound t on the number of channels the adversary can control.

In a SMT protocol, S has a message he wants to communicate privately and reliably to R. This means R receives the correct message with probability 1 and the adversary learns no new information about the message sent. Either S or R has to start the protocol, by sending something over the n channels (possibly an empty message). This is done in one round. In the second round, the direction is reversed, and the other party sends messages back, and so forth.

A SMT protocol is called perfect if both privacy and reliability hold with probability 1. In fact, most research done in the area of SMT concerns perfect SMT (PSMT) protocols. Since our AMD codes always have an nonzero error probability, their are not really useful to do PSMT. If $n > 3t + 1$, we can do PSMT in one round using Shamir's secret sharing scheme as follows:

In an initialization phase, S and R agree on parameters t, n a field K with $|K| > n$, and distinct, non-zero $x_1, \ldots, x_n \in K$ (the interpolation points for the secret sharing scheme). Now, S wants to send a message. What he does is compute a vector of shares $(s_1, \ldots, s_n)$ according to Shamir's secret sharing scheme. He then simply sends share s_i over channel i. R then receives a vector $(\tilde{s}_1, \ldots, \tilde{s}_n)$, possibly altered by the adversary (as indicated by the tildes). If R receives a full correct vector of shares, he simply executes the reconstruction phase of Shamir's scheme. If some shares are not or incorrectly received, he uses the Berlekamp/Welch error correction algorithm to obtain a full vector of shares. Since Shamir's scheme provides secrecy, our adversary learns nothing about the message m.

Now, suppose we want to communicate a message m in one round, but we can only guarantee $n > 2t + 1$. If we want to do PSMT with $n > 2t + 1$, we need at least two rounds. Therefore, given that we only have one round, we will have to accept a nonzero probability that the message does not arrive. There is a way of doing this using authentication codes, but it is rather involved. We will see that using AMD codes, this can be done in an elegant way as follows:

As before, R and S have an initialization phase to agree on parameters. Now, S wants to send a message m. He first uses a weakly secure AMD code to encode m to e. Note that a weakly secure code can be used, because the adversary has no knowledge about the source state. Then, he computes a vector of shares $(s_1, \ldots, s_n)$ for e, according to Shamir's secret sharing scheme. He simply sends share s_i over channel i. R then receives a vector $(\tilde{s}_1, \ldots, \tilde{s}_n)$, possibly altered by the adversary (as indicated by the tilde). Since $n > 2t + 1$, there is a set of at least $t + 1$ uncorrupted channels. So, what we do is simply invoke the reconstruction algorithm on each set of $t + 1$ shares, until no error is detected. For more information concerning reconstruction in robust secret sharing schemes, please see [7].

4 Weakly secure AMD codes based on difference sets

In this section, difference sets will be used to construct AMD codes. We will also list some important theorems about difference sets (all of these can be found in [2]).

4.1 Definition of difference sets

We will start with introducing the concept of a difference set. Let G be an abelian group of order v.

Definition 4.1.1. *A (v, k, λ)-difference set in G is a subset $D \subseteq G$ with $\#D = k$ such that every nonzero $g \in G$ occurs exactly λ times in the multiset $(x - y : x, y \in D)$ of differences.*

To make this definition more clear, we will look at a simple example. Consider $\{1, 2, 4\} \in \mathbb{Z}_7$. Then we have:

$$\begin{aligned} 2 - 1 &= 1 & \quad 1 - 4 &= -3 = 4 \\ 4 - 2 &= 2 & \quad 2 - 4 &= -2 = 5 \\ 4 - 1 &= 3 & \quad 1 - 2 &= -1 = 6 \end{aligned}$$

Therefore, since every nonzero $g \in G$ occurs exactly once as a difference of 2 elements in $\{1, 2, 4\}$, this set is a $(7, 3, 1)$ difference set in $\mathbb{Z}_7$. Other examples:
$\{0, 1, 3, 9\}$ in $\mathbb{Z}_{13}$ is a $(13, 4, 1)$ difference set.
$\{1, 3, 9, 5, 4\}$ in $\mathbb{Z}_{11}$ is a $(11, 5, 2)$ difference set.

An interesting fact to note is that v, k, and λ are related by the following formula:

$$\lambda(v - 1) = k(k - 1)$$

This is easily seen to be true, because there are $v - 1$ nonzero elements in G, and each of those elements occur λ times in the multiset $(x - y : x, y \in D)$, which has size $k(k - 1)$.

A difference set with $1 < k < v - 1$ is called nontrivial. A difference set with $\lambda = 1$ is usually called planar or simple. We now know enough to look at the construction of AMD codes from difference sets.

4.2 Constructing AMD codes from difference sets

The following construction is due to Ogata and Kurosawa [12]. Let D be a (v, k, λ)-difference set in G. Then, we can construct an AMD code as follows:

$$\begin{aligned} E : D &\rightarrow G \\ s &\mapsto s \end{aligned}$$

Theorem 4.2.1. *The above (k, v)-AMD code E is weakly ε-secure, where $\varepsilon = \frac{\lambda}{k}$.*

Proof. It is immediately clear that E is a (k, v)-AMD code. Now we have to argue that $\varepsilon = \frac{\lambda}{k}$. First, we fix a translation $\delta \neq 0 \in G$. Then, we count how many $s \in D$ there are such that $s + \delta \in D$. But by the properties of the difference set, δ occurs exactly λ times as a difference of two elements in D. Therefore, since δ is chosen independently from s, the probability that $s+\delta \in D$ is at most $\frac{\lambda}{k}$. □

Note that this is only a weakly secure AMD code, because if the source state is known to the adversary, he also knows the encoding and the code is broken.

In fact, in the proof of this theorem, we have never used the fact that every nonzero element occurs *exactly* λ times. This leads to the following interesting definition:

Definition 4.2.2. *A (v, k, λ)-bounded difference set in G is a subset $D \subseteq G$ with $\#D = k$ such that every nonzero $g \in G$ occurs at most λ times in the multiset $(x - y : x, y \in D)$ of differences.*

If we create a weakly secure AMD code from a bounded difference set, it has exactly the same properties as one created from a normal difference set. However, since a bounded difference set has relaxed combinatorial conditions, we could argue that there are probably more bounded difference sets than normal ones, and therefore more opportunities to create AMD codes from them. Unfortunately, all known important theorems about difference sets are about normal difference sets. Therefore, in the next subsections, we will focus on these instead of the possibly more promising bounded set approach.

4.3 Difference sets based on quadratic residues

Now that we know how to build AMD codes from difference sets, we would like to have a list of difference sets to choose from. One way of constructing difference sets is based on quadratic residues. The following theorem gives us a way of doing this:

Theorem 4.3.1. ***(Paley,Todd)*** *Let $q = 4n - 1$ be a prime power. Then the set D of nonzero squares in $\mathbb{F}_q$ is a $(4n - 1, 2n - 1, n - 1)$-difference set in the additive group of $\mathbb{F}_q$.*

Proof. It is well known that $|D| = 2n-1$. Since D is invariant under multiplication by elements of the set S of nonzero squares ($a^2b^2 = aabb = abab = (ab)^2$), the multiset M of differences from D has the same property. Also, M is invariant under multiplication by -1 (since this just turns $(a-b)$ into $(b-a)$). Since $q \equiv 3 \mod 4$, $-1 \notin S$ and every nonzero element of $\mathbb{F}_q$ is either in S or of the form $-s$ for $s \in S$. From this we can conclude that M is invariant under multiplication by all nonzero elements in $\mathbb{F}_q$ and so D is a $(4n-1, 2n-1, \lambda)$-difference set. Since $\lambda(v-1) = k(k-1)$ (this follows straight from the definition of a difference set), we know that $\lambda = \frac{(2n-1)(2n-2)}{4n-2} = n-1$. □

Using theorem 4.2.1, we can construct $(2n-1, 4n-1)$ weakly secure AMD codes with error probability $\varepsilon = \frac{n-1}{2n-1} \approx \frac{1}{2}$ from these kind of difference sets. Note that here there is no possibility to vary the error probability, which is clearly an unwanted situation.

4.4 Singer's theorem

Another way of constructing difference sets is by Singer's theorem. This theorem identifies hyperspaces in $\mathbb{P}^n(\mathbb{F}_q)$ with difference sets. The formal statement of this theorem and its proof require some combinatorics, so we will not list them here. Instead, the reader is invited to read about this in [2]. However, we will speak informally about this theorem and show how difference sets are constructed from it in practice. What happens is that hyperspaces in $\mathbb{P}^n(\mathbb{F}_q)$ are identified with difference sets with the following parameters:

$$v = \frac{q^{n+1}-1}{q-1}, \quad k = \frac{q^n-1}{q-1}, \quad \lambda = \frac{q^{n-1}-1}{q-1}.$$

Note that these difference sets are subsets of the cyclic group of order v. Now, we will talk about how to do this. Let ω be a primitive element of $\mathbb{F}_{q^{n+1}}$ and define $v = \frac{q^{n+1}-1}{q-1}$. Note that v is equal to the number of points in $\mathbb{P}^n(\mathbb{F}_q)$. This is true because the points in $\mathbb{P}^n(\mathbb{F}_q)$ are equivalence classes of points in $\mathbb{A}^{n+1}(\mathbb{F}_q)$, where two points are equivalent if for $\mathbf{x}, \mathbf{y} \in \mathbb{A}^{n+1}(\mathbb{F}_q) \neq \mathbf{0}$ holds that $\lambda\mathbf{x} = \mathbf{y}$ for some $\lambda \in \mathbb{F}_q \neq 0$. Since there are $q^{n+1}-1$ points in $\{\mathbb{A}^{n+1}(\mathbb{F}_q) \neq \mathbf{0}\}$, and every point is equivalent to $q-1$ other points, the number of points in $\mathbb{P}^n(\mathbb{F}_q)$ will become $\frac{q^{n+1}-1}{q-1}$. The cyclic multiplicative group $\langle\omega\rangle$ of $\mathbb{F}_{q^{n+1}}$ has a unique subgroup of order $q-1$, being

$$\langle\omega^v\rangle = \{\omega^0 = 1, \omega^v, \omega^{2v}, \dots, \omega^{(q-2)v}\}.$$

However, the multiplicative group of $\mathbb{F}_q$ also has order $q-1$, so we conclude that $\mathbb{F}_q = \{0, \omega^v, \omega^{2v}, \dots, \omega^{(q-2)v}\}$. Now two "vectors" ω^i and ω^j in $\mathbb{F}_{q^{n+1}}$, considered as vector space over $\mathbb{F}_q$, represent the same projective point if $\omega^i = \lambda\omega^j$ for $\lambda \in \mathbb{F}_q \neq 0$. This is the same as saying that $i \equiv j \mod v$. Thus, we now have a bijection between the projective points and the group $\mathbb{Z}_v$ of residue classes modulo v:

$$\begin{aligned} 0 \leftrightarrow x_0 &= \{0, \omega^0, \omega^v, \omega^{2v}, \dots, \omega^{(q-2)v}\} \\ 1 \leftrightarrow x_1 &= \{0, \omega^1, \omega^{v+1}, \omega^{2v+1}, \dots, \omega^{(q-2)v+1}\} \\ &\vdots \\ v-1 \leftrightarrow x_{v-1} &= \{0, \omega^{v-1}, \omega^{2v-1}, \omega^{3v-1}, \dots, \omega^{(q-1)v-1}\} \end{aligned}$$

Now, a difference set is obtained by taking any n-dimensional subspace S of $\mathbb{F}_{q^n+1}$, and let $D = \{i \in \mathbb{Z}_v : x_i \in S\}$.

Example:
Take $n = 2, q = 5$. We will construct a $(31, 6, 1)$-difference set. A zero ω of the polynomial $x^3 + x^2 + 2$ is a primitive element of $\mathbb{F}_{5^3}$ (because $\mathbb{F}_5(\omega)$ gives us an extension of $\mathbb{F}_5$ of degree 3 which is $\mathbb{F}_{5^3}$) and $1, \omega, \omega^2$ form a basis for $\mathbb{F}_{5^3}$ as a vector space over $\mathbb{F}_5$. The 124 nonzero elements of $\mathbb{F}_{5^3}$ fall into 31 cosets modulo the subgroup $\langle\omega^{31}\rangle = \{3, 4, 2, 1\} = \mathbb{F}_5 \setminus \{0\}$, each coset being the nonzero elements of a 1-dimensional subset. We will take $U = \text{span}\{1, \omega\}$ as our 2-dimensional subspace. Representatives of the projective points on U are $1, \omega, \omega + 1, \omega + 2, \omega + 3, \omega + 4$ and after some calculations we get that

$$\begin{aligned} 1 &= \omega^0 \\ \omega &= \omega^1 \\ \omega + 1 &= \omega^{29} \\ \omega + 2 &= \omega^{99} \\ \omega + 3 &= \omega^{80} \\ \omega + 4 &= \omega^{84} \end{aligned}$$

For example, $(\omega + 1) \cdot \omega^2 = \omega^3 + \omega^2 = -2$ (because of the given polynomial relation $\omega^3 + \omega^2 + 2 = 0$), and $-2 = 3 = \omega^{31}$, so $\omega + 1$ is $\frac{\omega^{31}}{\omega^2} = \omega^{29}$. Taking the exponents modulo 31, we obtain the following difference set in $\mathbb{Z}_{31}$: $\{0, 1, 29, 6, 18, 22\}$.

4.5 The Multiplier Theorem

Another important theorem concerning the existence of difference sets is the multiplier theorem. In order to prove this theorem, we first need to introduce group rings. Let R be a ring (commutative with one) and G a finite abelian group. The elements of the group ring $R[G]$ are formal sums

$$A = \sum_{g \in G} a_g x^g$$

where $a_g \in R$ for each $g \in G$. What this sum does is giving every element in G a corresponding element in R, so each mapping $G \to R$ corresponds to an element in the group ring. Addition and scalar multiplication are defined just like you would expect with sums:

$$\sum_{g \in G} a_g x^g + \sum_{g \in G} b_g x^g = \sum_{g \in G} (a_g + b_g) x^g,$$

$$c \sum_{g \in G} a_g x^g = \sum_{g \in G} (c a_g) x^g.$$

Multiplication is defined by

$$\sum_{g \in G} a_g x^g \sum_{g \in G} b_g x^g = \sum_{g \in G} \left(\sum_{h + h' = g} a_h b_{h'} \right) x^g.$$

Then, taking $x^0 \in R[G]$ as the unit element for multiplication (which we denote by 1), these definitions make $R[G]$ a ring. We will look mostly at group rings where the ring we are working with is that of the integers, i.e. $\mathbb{Z}[G]$. For a subset $A \subseteq G$, we define $A(x) \in \mathbb{Z}[G]$ by $A(x) = \sum_{g \in A} x^g$. Now, let $A, B \subseteq G$. Then:

$$A(x)B(x) = \sum_{g \in G} c_g x^g,$$

where c_g is the number of times g occurs in the multiset $(h + h' : h \in A, h' \in B)$. We define $A(x^{-1})$ as $\sum_{g \in A} x^{-g}$. We are now ready to "translate" the requirements on a difference set to an equation using group rings. Let G be a group of order v, and $D \subseteq G$ with $\#D = k$. Then D is a (v, k, λ)-difference set in G if and only if the following equation holds:

$$D(x)D(x^{-1}) = n + \lambda G(x),$$

with $n = k - \lambda$. This is not hard to see, since in a difference set, every nonzero element of G occurs λ times in the multiset of differences. However, when stated in group ring terms, we also get $0 \in G$ k times because we can take an element in D and take the difference with itself. Since $\lambda G(x)$ already gives us λ zeros, we need to add another n times zero to our equation, and $x^0 = 1$.

Now, we define what a multiplier of a difference set is. Let G be an abelian group and D a difference set in G. An automorphism α of G is said to be a multiplier of D if and only if $\alpha(D) = D + g$ for some $g \in G$. For example, 3 is a multiplier of the (13,4,1)-difference set $\{0, 2, 3, 7\} \in \mathbb{Z}_{13}$, because $3 \cdot \{0, 2, 3, 7\} = \{0, 6, 9, 8\} = \{7, 0, 3, 2\} + 6$. If $x \mapsto tx$ is a multiplier of a difference set $D \in G$, we say that t is a numerical multiplier of D. A fact which is not immediately clear is that all known cyclic difference sets have a nontrivial multiplier. We are now ready to state the famous multiplier theorem:

Theorem 4.5.1. ***(Hall, Ryser)*** *Let D be a (v, k, λ)-difference set in an abelian group G of order v. Let p be a prime, $p | k - \lambda$, $(p, v) = 1$, $p > \lambda$. Then p is a numerical multiplier of D.*

Note that as above, $n = k - \lambda$, to simplify our notation. We will use 2 lemmas to prove this theorem, just like the proof in [2].

Lemma 4.5.2. *Let p be a prime and $A \in \mathbb{Z}[G]$. Then*

$$(A(x))^p \equiv A(x^p) \mod p$$

Proof. We use induction on the number of nonzero coefficients of $A(x)$. The lemma trivially holds for $A(x) = 0$. Now if $A(x) = cx^g + B(x)$ with $(B(x))^p \equiv B(x^p) \mod p$, then

$$\begin{aligned} (A(x))^p = (cx^g + B(x))^p &\equiv (cx^g)^p + (B(x))^p \\ &= c^p x^{pg} + (B(x))^p \equiv cx^{pg} + B(x^p) = A(x^p) \end{aligned}$$

□

Lemma 4.5.3. *Let α be an automorphism of G and let D be a (v, k, λ)-difference set in G. Consider*

$$S(x) = D(x^\alpha)D(x^{-1}) - \lambda G(x).$$

Then α is a multiplier of D if and only if $S(x)$ has nonnegative coefficients.

Proof. $\Rightarrow$: If α is a multiplier of D, then we know from the previous that $\alpha(D) = D + g$, which we state in group ring terms as $D(x^\alpha) = x^g \cdot D(x)$ for some $g \in G$, and then we have

$$D(x^\alpha)D(x^{-1}) = x^g \cdot D(x)D(x^{-1}) = x^g(n + \lambda G(x)) = nx^g + \lambda G(x).$$

Note that $G(x)$ is invariant under multiplication with elements $g \in G$. So, assuming α is a multiplier of D, we have that $S(x) = nx^g$ for some $g \in G$ and thus has nonnegative coefficients.
$\Leftarrow$: We consider:

$$\begin{aligned} S(x)S(x^{-1}) &= (D(x^\alpha)D(x^{-1}) - \lambda G(x))(D(x^{-\alpha})D(x) - \lambda G(x)) \\ &= (n + \lambda G(x))^2 - 2\lambda k^2 G(x) + \lambda^2 v G(x), \end{aligned}$$

because $D(x^\alpha)D(x^{-1}) \cdot -\lambda G(x) = (-\lambda G(x)D(x^\alpha))D(x^{-1})$. Note that since for any $A \subseteq G : A(x)G(x) = |A|G(x)$, we have $G(x)^2 = vG(x)$. Now, we simplify the expression we found:

$$\begin{aligned} & (n + \lambda G(x))^2 - 2\lambda k^2 G(x) + \lambda^2 v G(x) \\ = \; & n^2 + 2\lambda(n + \lambda v - k^2)G(x) = n^2, \end{aligned}$$

because $n + \lambda v - k^2 = k - \lambda + \lambda v - k^2 = \lambda(v-1) + k - k^2 = \lambda(v-1) - k(k-1) = 0$. Suppose now that $S(x) = \sum_{g \in G} s_g x^g$ with nonnegative coefficients s_g. If $s_g > 0$ and $s_h > 0$ for $g, h \in G$, then the coefficient of x^{g-h} in $S(x)S(x^{-1}) = n^2$, is at least $s_g s_h$, which is larger than zero, and so $x^{g-h} = x^0$, so $g = h$. Now, we have shown that $S(x)$ having 2 distinct positive coefficients leads to a contradiction, and hence $S(x) = s_g x^g$. Since $S(x)S(x^{-1}) = n^2$, $s_g = n$ and we have shown that $S(x) = nx^g$. Now we need to argue that α is a multiplier:

$$\begin{aligned} nx^g &= D(x^\alpha)D(x^{-1}) - \lambda G(x) \\ nx^g + \lambda G(x) &= D(x^\alpha)D(x^{-1}) \\ x^g(n + \lambda G(x)) &= D(x^\alpha)D(x^{-1}) \\ x^g D(x)D(x^{-1}) &= D(x^\alpha)D(x^{-1}) \\ x^g D(x) &= D(x^\alpha), \end{aligned}$$

and we conclude that α is indeed a multiplier of D. □

We will now use these lemmas to prove the multiplier theorem.

Proof. Let $S(x) = D(x^p)D(x^{-1}) - \lambda G(x)$. By lemma 3.5.3, showing that $S(x)$ has nonnegative coefficients is enough to prove our theorem. By lemma 3.5.2,

$$\begin{aligned} D(x^p)D(x^{-1}) &\equiv (D(x))^p D(x^{-1}) = (D(x))^{p-1} D(x)D(x^{-1}) \\ &= (D(x))^{p-1} \cdot (n + \lambda G(x)) = n(D(x))^{p-1} + \lambda k^{p-1} G(x) \\ &\equiv \lambda G(x) \mod p, \end{aligned}$$

since $p|n$. Thus, the coefficients of $D(x^p)D(x^{-1})$, which are nonnegative, are all $\lambda \mod p$. Since $p > \lambda$, the coefficients of $D(x^p)D(x^{-1})$ are greater then or equal to λ. Therefore, $S(x) = D(x^p)D(x^{-1}) - \lambda G(x)$ has nonnegative coefficients and this implies that p is a multiplier. □

We will give a small example where we can see how the theorem is useful in practice.

Corollary 4.5.4. *For $\lambda = 1$, every prime divisor of n, and hence every divisor, is a multiplier of every $(n^2 + n + 1, n + 1, 1)$-difference set.*

Example: Let D be a hypothetical $(n^2+n+1, n+1, 1)$-difference set with $n \equiv 0 \mod 6$. Then both 2 and 3 are multipliers of D. Without loss of generality, we can take D to be fixed by multiplication by 2 and 3 (because we can always add a $g \in G$ to D such that this is true). Then for $x \in D$, $2x, 3x \in D$. This means $2x - x = 3x - 2x = x$, and for $x \neq 0$, these 2 are different ways of writing x as a difference, and this contradicts with $\lambda = 1$. Therefore, there are no $(n^2 + n + 1, n + 1, 1)$-difference sets with $n \equiv 0 \mod 6$.
We can see that the multiplier theorem is useful in excluding possible sets of parameters for difference sets. For example, it has been proven that for $n \leq 3600$, there exist no $(n^2 + n + 1, n + 1, 1)$-difference sets with n not a prime power.

There is much more combinatorial theory which tells us about the existence of certain difference sets, for example found in [2] and [3].

5 A nearly optimal weakly secure AMD code

We now present a nearly optimal weakly secure AMD code. It is based on the idea of the example in chapter 2.4, but with an interesting modification:

Let $\mathbb{F}$ be the finite field of size 3. Let $c, d : 1 \leq c \leq d$ be integers. Consider the AMD-code

$$\begin{aligned} E : \mathbb{F}^d &\longrightarrow \mathbb{F}^d \times \mathbb{F}^c \\ s &\longmapsto (s, (s^2)_c), \end{aligned}$$

where $(s^2)_c$ are the first c coordinates of s^2 written as a vector over $\mathbb{F}$. We compute this by first writing s as a vector over $\mathbb{F}$. Since $\mathbb{F}^d$ is a d-dimensional vector space over $\mathbb{F}$, we have a basis B which allows us to write elements in $\mathbb{F}^d$ as a vector with d entries in $\mathbb{F}$. We then multiply this vector by itself and get the first c coordinates. This works exactly the same as elements in $\mathbb{C}$, which we write as a 2-vector of elements in $\mathbb{R}$ (a+bi).

Theorem 5.0.5. *The above $(3^d, 3^{d+c})$-AMD code E is weakly ε-secure, where $\varepsilon = \frac{1}{3^c}$. Its effective overhead is $\bar{\omega}^*(\kappa, \ell) \leq \kappa + \log(3)$.*

Proof. It is clear that E is an $(3^d, 3^{d+c})$-AMD code. Now, we need to determine ε. Fix an arbitrary translation $(\Delta s, \Delta e) \in \mathbb{F}^d \times \mathbb{F}^c$ with $\Delta s \neq 0$. Now, we count the number of different s that solve the equation:

$$\begin{aligned} ((s + \Delta s)^2)_c &= (s^2)_c + \Delta e \\ (s^2 + 2s\Delta s + \Delta s^2)_c &= (s^2)_c + \Delta e \\ (s^2)_c + (2s\Delta s)_c + (\Delta s^2)_c &= (s^2)_c + \Delta e \\ (2s\Delta s)_c &= \Delta e - (\Delta s^2)_c \end{aligned}$$

In order to solve the last equality, we get a system of linear equations. To be precise, we will get c equations with d variables. If $c = d$, then we get the equation $2s\Delta s = \Delta e - \Delta s^2$, which has exactly one solution in $\mathbb{F}^d$. This means all c equations are linearly independent. Therefore, if $c < d$, we can choose $d - c$ variables as we wish and then find a solution to the system. Thus, there are 3^{d-c} different s that solve the system, out of a possible 3^d. This makes our code weakly $\frac{3^{d-c}}{3^d} = \frac{1}{3^c}$ secure. Now, we need to argue our overhead. We are given κ and ℓ. Then, we must have $q^d \geq 2^\ell \Rightarrow d\log(3) \geq \ell \Rightarrow d \geq \frac{\ell}{\log(3)}$. Therefore, we take d to be $\lceil \frac{\ell}{\log(3)} \rceil$. Similarly, since $3^{-c} \leq 2^{-\kappa}$, we have $-c\log(3) \leq -\kappa \Rightarrow c \geq \frac{\kappa}{\log(3)}$ and we take c to be $\lceil \frac{\kappa}{\log(3)} \rceil$. Then

$$\begin{aligned} \log(n) - \log(m) &= \log(3^{\lceil \frac{\ell}{\log(3)} \rceil + \lceil \frac{\kappa}{\log(3)} \rceil}) - \log(3^{\lceil \frac{\ell}{\log(3)} \rceil}) \\ &= (\lceil \frac{\ell}{\log(3)} \rceil + \lceil \frac{\kappa}{\log(3)} \rceil) \log(3) - \lceil \frac{\ell}{\log(3)} \rceil \log(3) \\ &= \lceil \frac{\kappa}{\log(3)} \rceil \log(3) \leq (\frac{\kappa}{\log(3)} + 1) \log(3) = \kappa + \log(3), \end{aligned}$$

and therefore we have

$$\bar{\omega}^*(\kappa, \ell) \leq \kappa + \log(3).$$

□

Note that except for a constant log(3), this overhead is essentially optimal.

6 Strongly secure AMD codes based on authentication codes

In this section, we will discuss strongly secure AMD codes based on authentication codes. Since authentication codes are well-studied in cryptology, this construction will gives us a large group of AMD codes.

6.1 Definition of an authentication code

First, we will need to introduce authentication codes.

Definition 6.1.1. *A (systematic) authentication code consists of non-empty finite sets $\mathcal{K}, \mathcal{M}, \mathcal{T}$, and a function*

$$f : \mathcal{K} \times \mathcal{M} \to \mathcal{T}.$$

The set $\mathcal{K}$ is called the key space, the set $\mathcal{M}$ is called the message or plain text space, and $\mathcal{T}$ is called the tag space.

As the name implies, authentication codes can be used for message authentication. Consider the following scenario: Alice wants to send a message to Bob. However, the only communication channel they share is not necessarily secure. Furthermore, Alice doesn't mind if the message is read by a third party, i.e. there is no privacy required. Alice and Bob will then agree on a secret key k chosen uniformly random from the key space $\mathcal{K}$ (say, when they have dinner at some restaurant and no-one can overhear them). If Alice then wants to send a message to Bob after their dinner, she calculates $f(k, m) = \tau$. She sends the pair (m, τ) to Bob. Bob receives the possibly altered message $(\tilde{m}, \tilde{\tau})$. He must check if $f(k, \tilde{m}) = \tilde{\tau}$. If this is true, he accepts the message as originating from Alice. Otherwise, he rejects it.

Note that we said the communication channel Alice and Bob share is not necessarily secure. Now, we have a look at what can go wrong. Suppose there is a third party involved, called Eve, who has access to the channel (but not to the secret key k!). She can then try and do two things:

1. Before Alice sends a message, Eve can try to make Bob accept a message of her own by sending him a pair (m, τ). This is called an impersonation attack.

2. After Alice sends a message, Eve can intercept this message and try to make Bob accept a different one. The is actually easier then the impersonation attack, because she now has knowledge of a message m with the corresponding tag τ. We call this a substitution attack.

We will have a look at the probabilities that these attacks succeed:

For each $m \in \mathcal{M}$, consider the random variable $F(m)$, which takes on the value $f(k, m)$ when k is chosen uniformly from the key space.

Definition 6.1.2. *The impersonation probability of an authentication code is given by*

$$P_I = \max_{m \in \mathcal{M}, \tau \in \mathcal{T}} Prob(F(m) = \tau)$$

Since our adversary does not know the secret key, k is chosen uniformly random from his point of view, and the stated probability is in fact his chance at impersonating Alice.

The substitution probability is a bit harder to formulate. Our adversary is given a message m and the corresponding tag τ. He now needs to construct a pair $(\tilde{m}, \tilde{\tau})$ with $f(k, \tilde{m}) = \tilde{\tau}$ and $m \neq \tilde{m}$. First, we fix an arbitrary $m \in \mathcal{M}$. For each $\tau \in \mathcal{T}$, the probability $\lambda(m, \tau)$ that our adversary succeeds, conditioned on $f(k, m) = \tau$, is

$$\lambda(m, \tau) = \max_{\tilde{m} \in \mathcal{M}, \tilde{\tau} \in \mathcal{T} : m \neq \tilde{m}} Prob(F(\tilde{m}) = \tilde{\tau} | F(m) = \tau).$$

Now, the probability $P_S(m)$ that our adversary succeeds given m is

$$P_S(m) = \sum_{\tau \in \mathcal{T}} \gamma(m, \tau) \cdot \lambda(m, \tau),$$

where $\gamma(m, \tau)$ is the probability that $\tau = f(k, m)$. This leads to the following definition:

Definition 6.1.3. *The substitution probability of an authentication code is given by*

$$P_S = \max_{m \in \mathcal{M}} P_S(m),$$

with $P_S(m)$ as given above.

Note that a lower bound on both probabilities is that $P_I, P_S \geq \frac{1}{|\mathcal{T}|}$, since our adversary can always just take a guess at the correct tag corresponding to any message.

6.2 Example of an authentication code

To make things a bit more explicit, we will now look at a concrete example of an authentication code. Given a finite field K, consider the following function:

$$\begin{array}{rcl} f : K^2 \times K & \rightarrow & K \\ ((k_0, k_1), m) & \mapsto & k_0 m + k_1 \end{array}$$

For this authentication code, we have $P_I = P_S = \frac{1}{|K|}$. We will argue these equalities for both probabilities.

- For impersonation, we are asked to produce a pair (m, τ) such that $f(k, m) = \tau$, without knowing k. There are $|K|^2$ possibilities for (k_0, k_1). Of these possibilities, there are $|K|$ pairs that solve the equation $k_0 m + k_1 = \tau$ (because for every choice of k_0, there is a unique k_1 that solves it). Therefore, $P_I = \frac{1}{|K|}$.

- For substitution, we are asked to produce a pair $(\tilde{m}, \tilde{\tau})$ given a correct pair (m, τ). So, our adversary needs to compute $\tilde{\tau}$ for a message $\tilde{m}$. To do this, he can add $\tilde{\tau} - \tau$ to the τ he already knows. This gives us

  $$\tilde{\tau} - \tau = k_0 \tilde{m} + k_1 - (k_0 m + k_1) = k_0(\tilde{m} - m)$$

  Since $\tilde{m} - m$ is known to the adversary, all he has to do is take a guess at k_0. Therefore, $P_S = \frac{1}{|K|}$.

6.3 Constructing AMD codes from authentication codes

Now, we are ready to look at AMD codes constructed from authentication codes.

Let $E' : \mathcal{S}' \to \mathcal{G}'$ be a *weakly* ε'-secure AMD code with $\mathcal{S}' = \mathcal{K}$. Consider the following AMD code:

$$\begin{array}{rcl} E : \mathcal{S} & \to & \mathcal{S} \times \mathcal{G}' \times \mathcal{T} \\ s & \mapsto & (s, E'(k), \tau) \end{array}$$

for $k \in_R \mathcal{K}$. The decoding function returns s if the secret key is decoded successfully and $f(k, s) = \tau$.

Theorem 6.3.1. *The above ($|\mathcal{S}|, |\mathcal{S}||\mathcal{G}'||\mathcal{T}|$)-AMD code E is ε-secure, where $\varepsilon = \varepsilon' + P_S$. If the underlying AMD code E' is systematic, then $\varepsilon = \max\{\varepsilon', P_S\}$.*

Proof. t is clear that E is an $(|\mathcal{S}|, |\mathcal{S}||\mathcal{G}'||\mathcal{T}|)$-AMD code. Now, we need to determine ε. Fix an arbitrary translation $(\Delta s, \Delta e', \Delta\tau) \in \mathcal{S} \times \mathcal{G}' \times \mathcal{T}$, with $\Delta s \neq 0$. Since E' is a ε'-weakly secure AMD code, the probability that the adversary can change the key of our authentication code without detection is ε'. He knows s, so this allows him to break the strongly secure code. Also, if $f(k, s + \Delta s) = \tau + \Delta\tau$, the AMD code is broken, and this happens with probability P_S. Thus, the probability that E is broken is at most $\varepsilon' + P_S$. If E' is systematic, the encoding e' has k as first component. Therefore, we can make a distinction between the cases $\Delta k = 0$ and $\Delta k \neq 0$.
$\Delta k = 0$: since the key is unchanged, we need to consider the probability that $f(k, s + \Delta s) = \tau + \Delta\tau$, which is at most P_S.
$\Delta k \neq 0$: k is encoded using an AMD code, so the probability that the adversary succeeds in changing k is at most ε'. □

Note that these codes look more promising than the earlier examples, which mostly had fixed error probability. This construction allows us to combine good weakly secure AMD codes and authentication codes to create strongly secure AMD codes. However, the effective overhead of these codes created with this construction will always equal or exceed $4\kappa - 2^{-2\kappa+1}$ (this bound is based on a bound on authentication codes [14]. For more information, refer to [1]). This means it is essentially always 2κ away from the lower bound on strongly secure codes. Therefore, we will have to look at other constructions to create better codes. However, we will first give an example of how this construction works in practice.

6.4 Example of a construction of a strongly secure AMD code

Using examples 2.4 and 6.2, we will now construct a strongly secure AMD code. Consider the following function:

$$\begin{array}{rcl} E : \mathcal{S} & \to & \mathcal{S} \times \mathcal{G}' \times \mathcal{T} \\ s & \mapsto & (s, (\mathbf{k}, \mathbf{k}^2), k_0 s + k_1) \end{array}$$

Theorem 6.4.1. *The above $(|\mathcal{S}|, |\mathcal{S}||\mathcal{G}'||\mathcal{T}|)$-AMD code E is ε-secure, where $\varepsilon = \frac{1}{|S|}$. Its effective overhead is $\bar{\omega}^*(\kappa, \ell) \geq \max\{5\kappa, 5\ell\}$.*

Proof. It is clear that E is an $(|\mathcal{S}|, |\mathcal{S}||\mathcal{G}'||\mathcal{T}|)$-AMD code. Now, we need to determine ε. Our weakly secure AMD code has k as the first component of its encoding, so $\varepsilon = \max\{\varepsilon', P_S\}$. $\varepsilon' = \frac{1}{\mathcal{K}} = \frac{1}{|\mathcal{S}|^2}$, and $P_S = \frac{1}{|\mathcal{S}|}$, so ε becomes $\frac{1}{|\mathcal{S}|}$. It remains to argue the size of the effective overhead. We know that $|\mathcal{T}| = |\mathcal{S}|$ from example 6.2. Also, $|\mathcal{K}| = |\mathcal{S}|^2$, and from example 2.4 we then get $|\mathcal{G}'| = |\mathcal{S}|^4$. So we now see that $\bar{\omega} = \log(n) - \log(m) = \log(|\mathcal{S}|^6) - \log(|\mathcal{S}|) = \log(|\mathcal{S}|^5)$. However, if we consider the effective overhead, we get parameters (κ, ℓ) and we need to find the code which satisfies $m \geq 2^\ell$ and $\varepsilon \leq 2^{-\kappa}$ with the lowest possible overhead. However, from the 2 given inequalities we get $|\mathcal{S}| \geq 2^\ell \Rightarrow \log(|\mathcal{S}|) \geq \ell$ and $|\mathcal{S}| \leq 2^{-\kappa} \Rightarrow \log(|\mathcal{S}|) \geq \kappa$. So, our effective overhead $\bar{\omega}^*$ is at least 5κ or 5ℓ, depending on which is bigger. □

Note that just like before, ℓ will usually be a lot bigger than κ, so this code is hardly efficient in practice.

7 Differential structures

In chapter 4, we have constructed AMD codes from difference sets. However, it became clear that using these constructions is not very efficient. Also, difference sets only produce weakly secure AMD codes because of the deterministic encoding. Part of this problem is caused by the ”smoothness” of difference sets. These sets have very nice combinatorial properties, which unfortunately allow us very little room to vary parameters to suit our needs. This leads to the idea of dropping certain combinatorial demands, hoping that this will improve the usability in our applications. This is an idea that has been used successfully in other areas of cryptology as well, e.g. authentication codes. Therefore, we will now look at a concept similar to difference sets, giving more ”freedom” but less combinatorial ”smoothness”.

7.1 Definition of a differential structure

The concept we will be using in the upcoming sections is that of a differential structure. We will first give a definition, and then look at a relevant parameter.

Let G be a finite group of order n. let $\mathcal{S}$ be a set of cardinality m, notated for simplicity as $\{1, \dots, m\}$. Let $V_1, \dots, V_m$ be disjoint, non-empty subsets of G.

Definition 7.1.1. *We call $(G, V_1, \dots, V_m)$ a differential structure.*

Now, let us look at our parameter. For any i, write t_i for the maximal overlap between any translation of V_i and the union of the other V_j's:

$$t_i = \max_{\delta \in G} |(V_i + \delta) \cap \bigcup_{j \neq i} V_j|.$$

7.2 Motivation for using differential structures

As promised, differential structures take care of some of the problems arising in constructing AMD codes from difference sets. In fact, differential structures immediately lead to AMD codes that are at strongly secure. To see how this works, consider the following code:

$$\begin{array}{rcl} E : \{1, \dots, m\} & \rightarrow & G \\ s & \mapsto & \tilde{s} \in_R V_s \end{array}$$

meaning $\tilde{s}$ is chosen uniformly random from V_s. It is again quite clear how our decoding function should work, i.e., we decode to s if $\tilde{s} \in V_s$ and output $\perp$ if $\tilde{s} \notin V_s$ for any s. This AMD code is based on a general differential structure. We have the following theorem about its error probability:

Theorem 7.2.1. *Differential structures give rise to strongly ε-secure (m, n)-AMD codes, where $\varepsilon = \max_i \frac{t_i}{|V_i|}$.*

Proof. Since $\#G = n$, it is clear that E is a (m, n)-AMD code. Now, we want to prove our bound on the error probability. First, we fix s. We can do this, because our ε should hold for all s. Let $\tilde{s}$ be the probabilistic encoding of s, and δ the "shift" chosen by our adversary, according to some distribution unknown to us but independent of $\tilde{s}$. Then, $\tilde{s} + \delta$ is uniformly distributed in $V_s + \delta$. Therefore, the probability that $\tilde{s} + \delta \in V_k$ for some $e \neq k$ is at most $\frac{t_s}{|V_s|}$. Thus, $\varepsilon = \max_i \frac{t_i}{|V_i|}$. □

Apart from the fact that differential structures lead to strongly secure AMD-codes, there is also more room to vary the error probability in relation to the size of the source space. This freedom allows us to construct codes which have an effective overhead that is closer to its lower bound. There are many ways of constructing differential structures, which will be the underlying principle in the upcoming sections.

8 Strongly secure AMD codes based on error correcting codes

In this section, we will look at a way of constructing AMD codes by using error correcting codes. First, we will need to define what an error correcting code is (definitions can be found in [13]). Then we will show the general construction to transform an ECC into an AMD code. Finally, an example will be given to help our intuition along.

8.1 Definition of an error correcting code

The type of error correcting codes we will be using is that of linear ECCs. Let $q = p^i$, $i \in \mathbb{Z}$ and p prime. Let $\mathbb{F}$ be the finite field of size q.

Definition 8.1.1. *A linear code C is a linear subspace of $\mathbb{F}^n$. If C has dimension k, then C becomes a (n, k)-code over $\mathbb{F}$.*

Definition 8.1.2. *A generator matrix G for a linear code C is a matrix whose rows consist of a set of basis vectors of C.*

It follows from the above definitions that a (n, k)-code has a generator matrix $G \in \mathbb{F}^{k\times n}$. The code now consists of all vectors $\mathbf{a}G$ with $a \in \mathbb{F}^k$. Vectors of this form are called codewords.

We will now define the Hamming-distance and weight, which will be useful later when discussing our construction.

Definition 8.1.3. *The Hamming-distance $d_H(\mathbf{x}, \mathbf{y})$ of two vectors $\mathbf{x}$ and $\mathbf{y}$ is*

$$d_H(\mathbf{x}, \mathbf{y}) = |i : 1 \leq i \leq n, x_i \neq y_i|$$

The weight $w(\mathbf{x})$ of $\mathbf{x}$ is $d_H(\mathbf{x}, \mathbf{0})$.

8.2 Constructing AMD codes from error correcting codes

With terminology given by the previous section, we are now ready to start constructing AMD codes.

Let $\mathbb{F}$ be a finite field of characteristic q, and let $G \in \mathbb{F}^{k\times n}$ be a generator matrix of a linear error correcting code C. We now consider the following AMD code:

$$\begin{aligned} E : \mathbb{F}^k &\rightarrow \mathbb{F}^k \times \mathbb{Z}_n \times \mathbb{F} \\ \mathbf{s} &\mapsto (\mathbf{s}, x, [\mathbf{s} \cdot G]_x) \end{aligned}$$

with $x \in_R \mathbb{Z}_n$, and where $[\mathbf{s} \cdot G]_x$ is equal to the x-th entry of the vector $\mathbf{s} \cdot G \in \mathbb{F}^n$ with the first entry being $x = 0$. In order to say something about the error probability of this code, we need to define some new concepts related to the definitions given in the previous section. Note that the final element of our encoding is a random entry from a codeword in C. Usually, we would use the Hamming distance to say something about the probability that this entry is the same in two codewords. However, since the entry we take can be influenced by our adversary (he can add a Δx to the original x), we also have to take rotations of the codeword into account. For example, if we look at $(2, 1, 0)$ and $(1, 0, 2)$, the Hamming distance between these vectors is three. But if we rotate the first vector one position to the left, we get $(1, 0, 2)$, which is exactly the same as the second vector, and hence the two now have Hamming distance zero.

First, for any $v \in \mathbb{F}^n$ let $M(v)$ be the maximum of the number of times a single element from $\mathbb{F}$ occurs as an entry of v: $M(v) := \max_{a \in \mathbb{F}} |i : v_i = a|$. For example, $M((4, 2, 1, 0, 0)) = 2$, since 0 occurs twice in the vector. Then, we can say that two codewords c and c' become "more similar" as $M(c - c')$ increases. Indeed, if we look at the two vectors for which we previously considered the Hamming distance, we now get that $M(c - c') = M((2, 1, 0) - (1, 0, 2)) = M((1, 1, 1)) = 3$, which is the maximum for vectors of length three. Now, we look at $\mu(C) := \max_{c \neq c' \in C} M(c - c')$. Clearly, μ determines the closest two codewords in C can get to each other.

Next, we define $\mathrm{rot}_t(c)$ of a codeword to be the codeword rotated t positions to the left. For example, $\mathrm{rot}_1((4, 2, 1, 0, 0) = (2, 1, 0, 0, 4)$ and $\mathrm{rot}_3((4, 2, 1, 0, 0)) = (0, 0, 4, 2, 1)$. Finally, we define the closure of $C \subseteq \mathbb{F}^n$ to be $cl(C) := \bigcup_{t \in \mathbb{Z}_n} \mathrm{rot}_t(C)$. These definitions are needed to determine how "good" an error code is in the setting of our construction. Now, we are ready to state a theorem concerning the error bound and overhead of our construction:

Theorem 8.2.1. *The above* (q^k, nq^{k+1})*-AMD code* E *is* ε*-secure, where* $\varepsilon = \frac{\mu(cl(C))}{n}$ *if* $\forall t \in \mathbb{Z}_n : rot_t(C) \cap C = \emptyset$*, and* $\varepsilon = 1$ *otherwise.*

Proof. It is clear that E is an (q^k, nq^{k+1})-AMD code. Now, we need to determine ε. We will now use the theory we developed in chapter 7. We simply define $V_{\mathbf{s}} := \{(\mathbf{s}, x, e) : e = [\mathbf{s} \cdot G]_x\}$ for all $\mathbf{s} \in \mathbb{F}^k$. It is immediately clear that $|V_{\mathbf{s}}| = n$. Thus, we only need to determine t_s. Fix an arbitrary translation $(\Delta\mathbf{s}, \Delta x, \Delta e) \in \mathbb{F}^k \times \mathbb{Z}_n \times \mathbb{F}$, with $\Delta\mathbf{s} \neq 0$. Now, we need to count the number of $x \in \mathbb{Z}_n$ which solve the following equation:

$$[\mathbf{s} \cdot G]_x + \Delta e = [(\mathbf{s} + \Delta\mathbf{s}) \cdot G]_{x+\Delta x}$$

Using our rot function, we can rewrite the right-hand side of the above equality as $[\mathrm{rot}_{\Delta x}((\mathbf{s} + \Delta\mathbf{s}) \cdot G)]_x$. Now, our new definitions come into play. Writing $c = \mathbf{s} \cdot G$ and $c' = (\mathbf{s} + \Delta\mathbf{s}) \cdot G$, we can now say that the number of x solving the equality is at most $M(c - \mathrm{rot}_{\Delta x}(c'))$, which is in turn upper bounded by $\mu(cl(C))$. Note that by assumption, $c \neq \mathrm{rot}_{\Delta x}(c')$. Using theorem 7.2.1, we can now state that $\varepsilon = \frac{\mu(cl(C))}{n}$. □

Now, we will look at an example of this construction.

8.3 Example of a construction of a strongly secure AMD code

Let $\mathbb{F}$ be a finite field of size q. Let G be the 1 by q matrix containing all elements of $\mathbb{F}$. Consider the AMD-code

$$\begin{aligned} E : \mathbb{F} &\rightarrow \mathbb{F} \times \mathbb{Z}_q \times \mathbb{F} \\ s &\mapsto (s, x, [s \cdot G]_x) \end{aligned}$$

with $x \in_R \mathbb{Z}_q$.

Theorem 8.3.1. *The above (q, q^3)-AMD code E is ε-secure, where $\varepsilon = \frac{1}{q}$. Its effective overhead is $\bar{\omega}^*(\kappa, \ell) \geq \max(2\kappa, 2\ell)$.*

Proof. It is clear that E is an (q, q^3)-AMD code. Now, we need to determine ε. Note that E is of the form described in 8.2. Therefore, we will only need to determine $\mu(cl(C))$ to find the error probability of our AMD code. Now, codewords in C are simply elements of $\mathbb{F}$. This means $M(v)$ will always be one, since one entry of $\mathbb{F}$ will occur once as an entry of v, and the other elements will occur zero times. Thus, $\mu(cl(C)) = \max_{c \neq c' \in C} M(c - c') = \max_{c \neq c' \in C} 1 = 1$. This leads to $\varepsilon = \frac{\mu(cl(C))}{n} = \frac{1}{q}$.
Regarding the overhead, we see that $\bar{\omega} = \log(n) - \log(m) = \log(q^3) - \log(q) = 2\log(q)$. However, if we consider the effective overhead, we get parameters (κ, ℓ) and we need to find the code which satisfies $m \geq 2^\ell$ and $\varepsilon \leq 2^{-\kappa}$ with the lowest possible overhead. However, from the 2 given inequalities we get $q \geq 2^\ell \Rightarrow \log(q) \geq \ell$ and $q \leq 2^{-\kappa} \Rightarrow \log(q) \geq \kappa$. So, our effective overhead $\bar{\omega}^*$ is at least 2κ or 2ℓ, depending on which is bigger. □

This means that the code is optimal if $\ell \leq \kappa$. However, this is rarely the case in practice. Furthermore, note that this is a very simple example of our construction. Using generator matrices which are more interesting will lead to more complicated proofs and possibly better AMD codes.

9 Strongly secure AMD codes based on polynomials

In this section, we will see examples of strongly secure AMD codes based on polynomials. The first example is listed here because its properties will be useful in discussing our multivariate example.

9.1 Univariate example

Let $\mathbb{F}$ be a finite field of size q, and let d be a positive integer. Consider the AMD code

$$\begin{aligned} E : \mathbb{F}^d &\rightarrow \mathbb{F}^d \times \mathbb{F} \times \mathbb{F} \\ \mathbf{s} = (s_1, \ldots, s_d) &\mapsto (\mathbf{s}, x, s_1 x + \ldots + s_d x^d + x^{d+2}) \end{aligned}$$

with $x \in_R \mathbb{F}$.

Theorem 9.1.1. *The above (q^d, q^{d+2})-AMD code E is ε-secure, where $\varepsilon = \frac{(d+1)}{q}$. Its effective overhead is bounded by $\bar{\omega}^*(\kappa, \ell) \leq 2\kappa + 2\log(\frac{\ell}{\kappa} + 2) + 2$.*

Proof. It is clear that E is an (q^d, q^{d+2})-AMD code. Now, we need to determine ε. Fix an arbitrary translation $(\Delta\mathbf{s}, \Delta x, \Delta e) \in \mathbb{F}^d \times \mathbb{F} \times \mathbb{F}$ with $\Delta\mathbf{s} \neq \mathbf{0}$. We write $s_i + \Delta s_i$ as s'_i and $x + \Delta x$ as x'. Now, we count the number of different x that solve the equation

$$s'_1 x' + \ldots + s'_d (x')^d + (x')^{d+2} = s_1 x + \ldots + s_d x^d + x^{d+2} + \Delta e$$

We can split this into two cases:
Case 1: $\Delta x = 0$.
The equation can then be simplified to become

$$s'_1 x + \ldots + s'_d x^d + x^{d+2} = s_1 x + \ldots + s_d x^d + x^{d+2} + \Delta e$$

We can see that the degree-$(d+2)$ term cancels out, and what remains is a polynomial equation $f(x) = 0$. Since $\Delta\mathbf{s} \neq \mathbf{0}$, this polynomial is not the zero polynomial, and since it has degree at most d, it will have at most d solutions.
Case 2: $\Delta x \neq 0$.
In this case, we can work out $(x')^{d+2} = (x + \Delta x)^{d+2} = x^{d+2} + \ldots + (\Delta x)^{d+2}$. We see that the degree-$(d+2)$ term cancels out again. This time however, we get a degree-$(d+1)$ term that is not cancelled, since the polynomial on the right-hand side does not have a degree-$(d+1)$ term. Therefore, our $f(x)$ will become a polynomial of degree $(d+1)$, and thus have at most $(d+1)$ solutions.
Since there are at most $d+1$ different x that solve our equation, our ε becomes $\frac{d+1}{q}$.
Now we need to prove our bound on the effective overhead. First, we fix our given κ and ℓ. Then, we choose $d = \lceil \frac{\ell}{\kappa} \rceil$ and $q = 2^{\lceil \kappa + \log(d+1) \rceil}$. Then, we get an AMD code with

$$m = q^d = q^{\lceil \frac{\ell}{\kappa} \rceil} \geq q^{\frac{\ell}{\kappa}} = 2^{\frac{\ell(\lceil \kappa + \log(d+1) \rceil)}{\kappa}} \geq 2^{\frac{\ell(\kappa + \log(d+1))}{\kappa}} \geq 2^{\ell}$$

and $\varepsilon = \frac{d+1}{q} = \frac{d+1}{2^{\lceil \kappa + \log(d+1) \rceil}} \leq \frac{d+1}{2^{\kappa + \log(d+1)}} = \frac{d+1}{2^{\kappa}(d+1)} = \frac{1}{2^{\kappa}}$.

The overhead of this code is

$$\begin{aligned}\bar{\omega} = \log q^{d+2} - \log q^{d} &= 2\log q = 2\log(2^{\lceil \kappa + \log(d+1) \rceil}) \leq 2\log(2^{\kappa+\log(d+1)+1}) \\ &= 2\kappa + 2\log(d+1) + 2 \leq 2\kappa + 2\log(\frac{\ell}{\kappa} + 2) + 2\end{aligned}$$

and then $\bar{\omega}^*(\kappa, \ell) \leq 2\kappa + 2\log(\frac{\ell}{\kappa} + 2) + 2$. □

9.2 Multivariate example

This example is based on the univariate construction of the previous section. Instead of using a polynomial with one variable, we now take one with r variables. We will also take d to be the maximal degree of the variables instead of $d+2$, since this makes the notation a lot easier.

Let $\mathbb{F}$ be a finite field of size q. Consider the AMD-code

$$\begin{aligned} E : \mathbb{F}^{(d+1)^r - r - 2} &\rightarrow \mathbb{F}^{(d+1)^r - r - 2} \times \mathbb{F}^{r+1} \\ \mathbf{s} &\mapsto (\mathbf{s}, x_1, \ldots, x_r, f(x)) \end{aligned}$$

with $\forall i : x_i \in_R \mathbb{F}$ and f a polynomial constructed by identifying $s \in \mathbb{F}^{(d+1)^r - r - 2}$ in a fixed, default way with an r-variate polynomial $f(X_1, \ldots, X_R)$ of degree d in each variable, and thus of total degree rd. We want f to have 1 as the coefficient of the degree-(rd) term, no degree-(*rd-1*) terms, and no constant coefficient. In order to get a feeling of how this works, consider an example with $d = 2$ and $r = 2$. Our polynomial then becomes

$$x_1^2 x_2^2 + s_1 x_1^2 + s_2 x_1 + s_3 x_2^2 + s_4 x_2 + s_5 x_1 x_2,$$

and indeed, we have identified $\mathbf{s} \in \mathbb{F}^5$ with a polynomial that has the required properties. It immediately follows that f has total degree $rd = 4$.

Theorem 9.2.1. *The above* $(q^{(d+1)^r - r - 2}, q^{(d+1)^r - 1})$*-AMD code* E *is* ε*-secure, where* $\varepsilon = \frac{rd-1}{q}$*. Its effective overhead is bounded by* $\bar{\omega}^*(\kappa, \ell) \leq (r+1)\kappa + (r+1)\log(r(\sqrt[r]{\frac{\ell}{\kappa} + r + 2}) - 1) + r + 1$.

Proof. It is clear that E is an $(q^{(d+1)^r - r - 2}, q^{(d+1)^r - 1})$-AMD code. Now, we need to determine ε. Fix an arbitrary translation $(\Delta\mathbf{s}, \Delta x_1, \ldots, \Delta x_r, \Delta e) \in \mathbb{F}^{(d+1)^r - r - 2} \times \mathbb{F}^{r+1}$ with $\Delta\mathbf{s} \neq \mathbf{0}$. Now, we need to count the number of $\mathbf{x}$ that solve the equation

$$f(\mathbf{x}) + \Delta e = f'(\mathbf{x}'),$$

where $\mathbf{x} = x_1, \ldots, x_r$, $\mathbf{x}' = x_1 + \Delta x_1, \ldots, x_r + \Delta x_r$ and f' the polynomial defined by $\mathbf{s} + \Delta\mathbf{s}$. Like the univariate example, we will split this into two cases:
Case 1: $\Delta x_i = 0$ for all $i \in \{1, \ldots, r\}$.
The equation we need to solve then simplifies to

$$f(\mathbf{x}) + \Delta e = f'(\mathbf{x}) \Rightarrow f(\mathbf{x}) - f'(\mathbf{x}) + \Delta e = 0.$$

Since $\Delta s \neq 0$, we know that $f(\mathbf{x}) - f'(\mathbf{x}) + \Delta e$ is not the zero polynomial. Since both f and f' have a term $x_1^d x_2^d \ldots x_r^d$ with coefficient 1, this term is cancelled out. Thus, we want to find the probability that a polynomial of total degree at most $rd - 2$ vanishes. For this, we use the following lemma:

Lemma 9.2.2. *(**Schwarz** [4]) For any nonzero polynomial $f \in \mathbb{F}[X_1, \ldots, X_r]$ of total degree at most d, the probability that f vanishes on a uniformly distributed tuple $(x_1, \ldots, x_r) \in \mathbb{F}^r$ is at most $\frac{d}{|\mathbb{F}|}$.*

Therefore, by Schwarz' lemma, the probability that adversary succeeds in Case 1 is at most $\frac{rd-2}{q}$.
Case 2: $\Delta x_i \neq 0$ for one or more $i \in \{1, \ldots, r\}$.
First, we examine the case in which there is exactly one $i \in \{1, \ldots, r\}$. In this case, we can expand $x_1^d x_2^d \ldots (x_i + \Delta x_i)^d \ldots x_r^d$ which becomes

$$x_1^d x_2^d \ldots x_r^d \quad + \quad x_1^d x_2^d \ldots d x_i^{d-1} \Delta x_i \ldots x_r^d \quad + \quad \ldots \quad + \\ x_1^d x_2^d \ldots d x_i \Delta x_i^{d-1} \ldots x_r^d \quad + \quad x_1^d x_2^d \ldots \Delta x_i^d \ldots x_r^d.$$

This might look a bit difficult, but the only thing we need to note is that there is now a degree-(rd) term with coefficient 1 and a degree-(*rd-1*) term. Since f also has a degree-(rd) term with coefficient 1, these cancel out again. The important fact however is that f has no degree-(*rd-1*) terms. Therefore, the equation we need to solve becomes a polynomial of degree at most $rd - 1$ (it is even exactly $rd - 1$ in this case, but that does not matter).
By Schwarz' lemma we get an error probability of at most $\frac{rd-1}{q}$. If there are more i for which $\Delta x_i \neq 0$, this procedure can be repeated with each variable, obtaining the same result in the end. Note that every time this procedure is repeated, we end with a degree-(rd) term with coefficient 1 and one more degree-(*rd-1*) term. The degree-(*rd-1*) terms will never cancel out, since they have power $d-1$ in different variables. This establishes our bound on the error probability of the code.
Now we need to prove our bound on the effective overhead. First, we fix our given κ and ℓ. Then, we choose $d = \lceil \sqrt[r]{\frac{\ell}{\kappa} + r + 2} \rceil - 1$ and $q = 2^{\lceil \kappa + \log(rd-1) \rceil}$. Then, we get an AMD code with

$$\begin{aligned} m = q^{(d+1)^r - r - 2} &= q^{(\lceil \sqrt[r]{\frac{\ell}{\kappa}+r+2} \rceil)^r - r - 2} \geq q^{(\sqrt[r]{\frac{\ell}{\kappa}+r+2})^r - r - 2} = q^{\frac{\ell}{\kappa}} \\ &= 2^{\frac{\ell(\lceil \kappa + \log(rd-1) \rceil)}{\kappa}} \geq 2^{\frac{\ell(\kappa + \log(rd-1))}{\kappa}} \geq 2^{\ell} \end{aligned}$$

and $\varepsilon = \frac{rd-1}{q} = \frac{rd-1}{2^{\lceil \kappa + \log(d+1) \rceil}} \leq \frac{rd-1}{2^{\kappa + \log(rd-1)}} = \frac{rd-1}{2^{\kappa}(rd-1)} = \frac{1}{2^{\kappa}}$.

The overhead of this code is

$$\begin{aligned} \bar{\omega} &= \log q^{((d+1)^r - r - 2) + (r+1)} - \log q^{(d+1)^r - r - 2} = (r+1) \log q \\ &= (r+1) \log 2^{\lceil \kappa + \log(rd-1) \rceil} \leq (r+1) \log 2^{\kappa + \log(rd-1) + 1} \\ &= (r+1)\kappa + (r+1)\log(rd-1) + (r+1) \\ &= (r+1)\kappa + (r+1)\log(r(\lceil \sqrt[r]{\frac{\ell}{\kappa} + r + 2} \rceil - 1) - 1) + (r+1) \\ &\leq (r+1)\kappa + (r+1)\log(r(\sqrt[r]{\frac{\ell}{\kappa} + r + 2}) - 1) + r + 1 \end{aligned}$$

and thus $\bar{\omega}^*(\kappa, \ell) \leq (r+1)\kappa + (r+1)\log(r(\sqrt[r]{\frac{\ell}{\kappa} + r + 2}) - 1) + r + 1$. □

Note that if we fill in $r = 1$ in the effective overhead above, we get exactly the same bound as in the univariate example.

10 Conclusion

As we have seen, AMD codes can be applied in several areas of cryptology. They are a very elegant way to do things such as robust secret sharing or non-perfect secure message transmission. Not only that, but many results known from cryptology and combinatorics can be used to construct AMD codes. We have seen constructions based on difference sets, authentication codes, error correcting codes and polynomials. Therefore, it is safe to say that AMD codes are a very interesting subject to study.

Luckily, there are still a lot of challenges left. For example, one could try to find a polynomial construction that is closer to optimal than the one given in the last chapter. Although the construction using authentication codes has proved rather unsuccessful, there might be error correcting codes that are very well suited for our application. Also, different constructions might be possible, for example using elliptic curves or bounded difference sets. Other applications for AMD codes could also be researched.

Finally, I'd like to thank both Prof. Dr. R.J.F. Cramer and Dr. S. Fehr at CWI, Amsterdam, for their assistance in writing this thesis. It could not have been done without them.

References

[1] R. Cramer, S. Fehr and C. Padró: *Combinatorial Codes for Detection of Algebraic Manipulation and Their Applications.* Manuscript, 2006.

[2] J.H. van Lint & R.M. Wilson: *A course in Combinatorics*, 2nd edition, Cambridge University Press, 2001

[3] M. Hall, jr.: *Combinatorial Theory*, Blaisdell Publishing Company, 1967

[4] J.T. Schwarz: *Fast probabilistic algorithms for verification of polynomial identities*, Journal of the ACM, 27(4), 1980.

[5] A. Shamir: *How to share a secret*, Communications of the Association for Computer Machinery, 22(11), 1979.

[6] S. Cabello, C. Padró, and G. Sáez. Secret sharing schemes with detection of cheaters for a general access structure. Designs, Codes and Cryptography 25 (2002) 175-188. Earlier version in Proceedings 12th International Symposium on Fundamentals of Computation Theory (FCT), volume 1233 of Lecture Notes in Computer Science. Springer, 1999.

[7] R. Cramer, I. B. Damgård, and S. Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In Advances in Cryptology – CRYPTO 01, volume 2139 of Lecture Notes in Computer Science. Springer, 2001.

[8] M. Tompa and H. Woll. How to share a secret with cheaters. Journal of Cryptology, 1(3), 1988.

[9] C. Blundo and A. De Santis. Lower bounds for robust secret sharing schemes. Information Processing Letters, 63(6), 1997

[10] Y. Desmedt and Y. Wang. Perfectly secure message transmission revisited. In Advances in Cryptology – EUROCRYPT 92, volume 658 of Lecture Notes in Computer Science. Springer, 1992.

[11] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. Journal of the ACM, 40(1), 1993.

[12] W. Ogata and K. Kurosawa. Optimum secret sharing scheme secure against cheating. In Advances in Cryptology – EUROCRYPT 96, volume 1070 of Lecture Notes in Computer Science. Springer, 1996.

[13] J.H. van Lint: *Inleiding in de coderingstheorie*, Mathematisch Centrum Amsterdam, 1976

[14] G.J. Simmons. Authentication theory/Coding Theory. In Advances in Cryptology – CRYPTO 84, volume 196 of Lecture Notes in Computer Science. Springer, 1985.