CS 7810 Graduate Cryptography November 29, 2017

Lecture 23: (Leveled) Fully Homomorphic Encryption

Lecturer: Daniel Wichs *Scribe: Jack Doerner*

1 Topics Covered

2. Definitions
3. Remembering Regev Encryption
4. The Short Integers Solution Problem
5. An Aside: CRHFs from LWE
6. The Gadget Matrix
7. The GSW Leveled FHE Scheme
8. The Security of GSW

2 Definitions

Definition 1 (Collision Resistant Hash Function)**.**

A family of functions

$$H_{\{0,1\}^n} : \{0,1\}^{\ell(n)} \mapsto \{0,1\}^n$$

with $\ell(n) = \text{poly}(n)$ is a Collision Resistant Hash Function Family (CRHF Family) if and only if

1. $\forall k \in \{0,1\}^n$, $\forall x \in \{0,1\}^{\ell(n)}$, $H_k(x)$ is computable in polnomial time.
2. It is not possible to efficiently find two inputs x and y such that the outputs of a randomly selected member of the family on x and y are equal. That is, for all PPT algorithms $\mathcal{A}$

$$\Pr\left[H_k(x) = H_k(y) : k \leftarrow U_n, (x,y) \leftarrow \mathcal{A}(k)\right] \leq \text{negl}(n)$$

Definition 2 (Learning With Errors Assumption)**.**

The Learning With Errors Assumption (LWE) is a cryptographic assumption over vectors and learning algorithms. Given a PPT algorithm

$$(\vec{s}, n, q, m, \chi) \leftarrow \mathsf{LWEGen}(1^n)$$

which takes a security parameter 1^n and produces a vector $\vec{s} \in \mathbb{Z}_q^n$ along with a distribution χ and a sample count m in poly(n), the Learning With Errors Assumption comes in two flavors, which imply one another:

1. The Decisional Learning With Errors Assumption (DLWE) asserts that if

$$(\vec{s}, n, q, m, \chi) \leftarrow \mathsf{LWEGen}\left(1^n\right), A \leftarrow \mathbb{Z}_q^{n \times m}, \vec{e} \leftarrow \chi^m, \vec{b} \leftarrow \mathbb{Z}_q^m$$

   then

$$(A, \vec{s}A + \vec{e}) \stackrel{c}{\equiv} \left(A, \vec{b}\right)$$

   In other words, m random linear combinations of the elements of $\vec{s}$ with errors $\vec{e}$ cannot be efficiently distinguished m uniform samples (given by $\vec{b}$), even when the matrix A that generated the combinations is known.

2. The Search Learning With Errors Assumption (SLWE) asserts that for all PPT algorithms $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} \mathcal{A}(A, \vec{s}A + \vec{e}) = \vec{s} : \\ \quad (\vec{s}, n, q, m, \chi) \leftarrow \mathsf{LWEGen}\left(1^n\right), \\ \quad A \leftarrow \mathbb{Z}_q^{n \times m}, \vec{e} \leftarrow \chi^m \end{array}\right] \leq \text{negl}(n)$$

   In other words, given m random linear combinations of the elements of $\vec{s}$ with errors $\vec{e}$ and the matrix A used to generate the combinations, $\vec{s}$ cannot be efficiently calculated.

Definition 3 (Public Key Encryption with IND-CPA Security)**.**

A Public Key Encryption (PKE) is a tuple of PPT algorithms, $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

1. Given a security parameter n, the Gen algorithm outputs a public key/private key pair: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$
2. Given a public key pk and a message m in the message space $\mathcal{M}$, the Enc algorithm outputs an encryption c: $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$
3. Given a private key sk and an encryption c, the Dec algorithm deterministically outputs a message: $m' \coloneqq \mathsf{Dec}_{\mathsf{sk}}(c)$

A Public Key Encryption Scheme with IND-CPA Security must conform to two properties:

1. (Correctness) With overwhelmingly high probability, all valid encryptions of a message must decrypt to the same message. Formally, we require that over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ and all messages m in the message space,

$$\Pr\left[\begin{array}{l} \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = m : \\ \quad m \leftarrow \mathcal{M}, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n) \end{array}\right] \geq 1 - \text{negl}(n)$$

2. (Indistinguishability under Chosen Plaintext Attacks (IND-CPA Security)) With overwhelmingly high probability, all ciphertexts must be indistinguishable to polynomial time adversaries that have access to the public key. Formally, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ comprising two algorithms which pass the state s between them, we insist that

$$\left| \Pr \left[\begin{array}{l} \mathcal{A}_2\left(c, s\right) = b : \\ \quad (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n), (m_0, m_1, s) \leftarrow \mathcal{A}_1\left(\mathsf{pk}\right), \\ \quad b \leftarrow U_1, c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b) \end{array} \right] - \frac{1}{2} \right| \leq \mathrm{negl}(n)$$

That is, the adversary receives the public key and, after performing a polynomial number of encryptions of its choice, it produces two challenge messages. The game encrypts one of these two messages at random and returns the ciphertext c to the adversary, which may then perform a polynomial number of encryptions again before guessing which of its messages was encrypted to produce c. The adversary must be correct with negligible advantage, relative to guessing at random.

3 Remembering Regev Encryption

Recall first the Regev Encryption Scheme:

Algorithm 1. $\mathsf{RegevGen}(n, m, \chi)$**:**

1. $\vec{s} \leftarrow \mathbb{Z}_q^n$
2. $A \leftarrow \mathbb{Z}_q^{n \times m}$
3. $\vec{e} \leftarrow \chi^m$
4. $\mathsf{pk} \coloneqq (A, \vec{b} = \vec{s}A + \vec{e})$
5. $\mathsf{sk} \coloneqq \vec{s}$
6. output $(\mathsf{pk}, \mathsf{sk})$

Algorithm 2. $\mathsf{RegevEnc}_{\mathsf{pk}=(A,\vec{b})}(x \in \{0,1\})$**:**

1. $\vec{r} \leftarrow \{0,1\}^m$
2. $\vec{u} \coloneqq A\vec{r}$
3. $\vec{v} \coloneqq \left\langle \vec{b}, \vec{r} \right\rangle + x \left\lceil \frac{q}{2} \right\rfloor$
4. output $\mathsf{ct} = (\vec{u}, \vec{v})$

Algorithm 3. $\mathsf{RegevDec}_{\mathsf{sk}=\vec{s}}(\mathsf{ct} = (\vec{u}, \vec{v}))$**:**

1. output $\lceil \vec{v} - \langle \vec{s}, \vec{u} \rangle \rfloor$

Notice with respect to the encryption algorithm that if an adversary is given $\vec{u}$, it should be hard for that adversary to recover $\vec{r}$, or in fact any smallish vector $\vec{r'}$ s.t. $A\vec{r'} = \vec{u}$. Otherwise, the adversary can use $\vec{b}$, which is part of the public key, to compute $\left\langle \vec{b}, \vec{r} \right\rangle$, which can be subtracted from $\vec{v}$ to recover x. Thus the problem of breaking Regev encryption can be reduced to the problem of finding such a vector. This is known as the Short Integer Solution (SIS) problem.

4 The Short Integer Solutions Problem

The Short Integer Solution Problem is a simple problem over matrices and vectors which must be hard to solve if LWE is hard. It directly models the security of Regev Encryption, specified above. The problem is of the following form: suppose there exists some public matrix $A \leftarrow \mathbb{Z}_q^{n \times m}$. Given a vector $\vec{u} \in \mathbb{Z}_q^n$, the problem is to find a vector $\vec{r'} \in \mathbb{Z}_q^m$ such that $A\vec{r} = \vec{u}$ and $\|\vec{r}\|_\infty \leq \gamma$ (for some smallish γ). Alternately, the problem can be stated in the following way: find a vector $\vec{r} \neq \vec{0}$ such that $A\vec{r} = \vec{0}$ and $\|\vec{r}\|_\infty \leq \gamma$ for $\gamma \ll q/(2m)$. This is conjectured to be computationally infeasible, and indeed, if it is not, it can be used to break LWE.

Theorem 1. *If the DLWE problem is hard, then the SIS problem must be as well.*

Proof. Suppose this were not the case. Specifically, suppose DLWE is hard to solve, and there were some adversary $\mathcal{A}$ who could recover $\vec{r} \neq \vec{0}$ given $A \leftarrow \mathbb{Z}_q^{n \times m}$ such that $A\vec{r} = \vec{0}$ and $\|\vec{r}\|_\infty \leq \gamma$. Hardness of the DLWE problem implies that

$$\left\{ \left(A, \vec{b} = \vec{s}A + \vec{e} \right) : A \leftarrow \mathbb{Z}_q^{n \times m}, \vec{s} \leftarrow \mathbb{Z}_q^n, \vec{e} \leftarrow \chi^m \right\} \stackrel{c}{\equiv} \left\{ \left(A, \vec{b} \right) : A \leftarrow \mathbb{Z}_q^{n \times m}, \vec{b} \leftarrow \mathbb{Z}_q^m \right\}$$

We could use $\vec{r}$ recovered by $\mathcal{A}$ to break this property by computing $\left\langle \vec{b}, \vec{r} \right\rangle$. If $\vec{b} = \vec{s}A + \vec{e}$ then we have

$$\begin{aligned} \left\langle \vec{b}, \vec{r} \right\rangle &= \langle \vec{s}A + \vec{e}, \vec{r} \rangle \\ &= \left\langle \vec{s}A\vec{r} + \vec{e}\vec{r}, \vec{1} \right\rangle \\ &= \left\langle \vec{0} + \vec{e}\vec{r}, \vec{1} \right\rangle \\ &\leq m\beta\gamma \end{aligned}$$

where β is the bound on χ. On the other hand, if $\vec{b} \leftarrow \mathbb{Z}_q^m$, then $\left\langle \vec{b}, \vec{r} \right\rangle$ is uniform over $\mathbb{Z}_q$ regardless of the value of $\vec{r}$. Thus, if $m\beta\gamma < q/2$, which is true if both β and γ are sufficiently small, then we can distinguish between the two distributions with non-negligible probability. This contradicts our assumption that DLWE is hard to solve, and so if DLWE is hard to solve, then SIS must also be. □

5 An Aside: CRHFs from LWE

We define the Hash function $H_{\mathbb{Z}_q^{n\times m}} : \mathbb{Z}_\gamma^m \mapsto \mathbb{Z}_q^n$ with $m > n\log q$ and $m = \text{poly}(n)$ to be calculated as

$$H_A\left(\vec{r} \in \left[\frac{\gamma}{2}, -\frac{\gamma}{2}\right]^m\right) = A\vec{r}$$

where $A \leftarrow \mathbb{Z}_q^{n\times m}$. Suppose that you could find a collision; specifically, $\vec{r}, \vec{r'}$ such that $\vec{r} \neq \vec{r'} \wedge A\vec{r} = A\vec{r'}$. This implies that $A(\vec{r} - \vec{r'}) = \vec{0}$ and $\|\vec{r} - \vec{r'}\|_\infty \leq \gamma$, which implies that you can break SIS and therefore LWE. Therefore, this must be a collision resistant hash function if LWE is hard.

6 The Gadget Matrix

We have shown that solving SIS is hard in the average case, but in some specific cases it is easy. The case in which we are interested is known as the Gadget Matrix:

$$G = \left[\begin{array}{cccccccccccc|cc} 2^0 & 2^1 & \cdots & 2^{\lfloor\log q\rfloor} & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\ 0 & 0 & \cdots & 0 & 2^0 & 2^1 & \cdots & 2^{\lfloor\log q\rfloor} & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 2^0 & 2^1 & \cdots & 2^{\lfloor\log q\rfloor} & 0 & \cdots \end{array}\right]$$

which is in $\mathbb{Z}_q^{n\times n\log q}\|0^{n\times(m-n\log q)}$. Note that this matrix effectively reconstructs values from vectors of bits. Consequently, solving the SIS problem relative to it is easy. Given any $\vec{u} \in \mathbb{Z}_q^n$, we can always easily find $\vec{r} \in \{0,1\}^m$ such that $G\vec{r} = \vec{u}$ simply by computing $\vec{r} \coloneqq \mathsf{Bits}(\vec{u})\|0$. We will make a slight abuse of notation to call the function that calculates this value $G^{-1}(\cdot)$, such that $GG^{-1}(\vec{u}) = \vec{u}$.

7 The GSW Leveled FHE Scheme

A Homomorphic Encryption (HE) Scheme is a variant of public-key encryption wherein the ciphertexts have some sort of homomorphism. That is, given two ciphertexts, C_1, C_2, there is some operator that can combine those ciphertexts *without knowledge of the secret key* such that the combined ciphertext decrypts to a well-defined combination of the original values. For example, there may exist some operation $\mathsf{HomomorphicAdd}$ such that $\mathsf{Dec}_{\mathsf{sk}}\left(\mathsf{HomomorphicAdd}(C_1, C_2)\right) = x_1 + x_2$, where x_1 and x_2 are the decryptions of C_1 and C_2 individually. Note that homomorphic encryption schemes are malleable by definition, and therefore they cannot have ciphertext indistinguishability under adaptive ciphertext attacks (IND-CCA2). A *Fully* Homomorphic Encryption (FHE) Scheme is an encryption scheme with homomorphisms sufficiently powerful to compute any function upon ciphertexts. For boolean circuits, this typically means XOR and AND, and for arithmetic circuits, it typically means addition and multiplication. A *Leveled* FHE scheme is an encryption scheme for which any function can be computed over the ciphertexts, so long as that function can be computed by a circuit of a particular, predetermined depth. What follows is

the scheme of Gentry, Sahai, and Waters (GSW), which is somewhat similar to the Regev encryption scheme. Since we will not show how to boostrap (i.e. reduce the error magnitude of a ciphertext), the scheme is a Leveled FHE as described. We begin with key generation:

Algorithm 4. $\mathsf{GSWGen}(n, m, \chi)$**:**

1. $\left(\left(A, \vec{b}\right), \vec{s}\right) \leftarrow \mathsf{RegevGen}(n, m-1, \chi)$
2. $\mathsf{pk} = \bar{A} := \begin{bmatrix} A \\ \vec{b} \end{bmatrix}$
3. $\mathsf{sk} = \vec{t} := -\vec{s} \| 1$
4. output $(\mathsf{pk}, \mathsf{sk})$

Note that GSWGen is effectively identical to $\mathsf{RegevGen}$, except that the output is in a slightly different format such that $\vec{t}\bar{A} = -\vec{s}A + \vec{b} = \vec{e}$. Next, we specify encryption:

Algorithm 5. $\mathsf{GSWEnc}_{\mathsf{pk}=(\bar{A})}(x \in \{0,1\})$**:**

1. $R \leftarrow \{0,1\}^{m \times m}$
2. $C := \bar{A}R + x \cdot G$
3. output C

In the above, G is the gadget matrix. Notice that

$$\begin{aligned} \vec{t}C &= \vec{t}\bar{A}R + x \cdot \vec{t}G \\ &= \vec{e}R + x \cdot \vec{t}G \\ &= x \cdot \vec{t}G + \vec{e^*} \end{aligned}$$

where $\|\vec{e^*}\|_\infty \leq m\beta$, and β is the bound on χ. We say that C is a γ-noisy encryption of x if and only if $\vec{t}C = x \cdot \vec{t}G + \vec{e^*}$ s.t. $\|\vec{e^*}\|_\infty \leq \gamma$

Algorithm 6. $\mathsf{GSWDec}_{\mathsf{sk}=\vec{t}}(C)$**:**

1. output $\vec{t}C \cdot G^{-1}\left(\vec{0} \| \left\lfloor \frac{q}{2} \right\rfloor\right)$

Note that

$$\begin{aligned}
&\vec{t}C \cdot G^{-1}\left(\vec{0}\| \left\lfloor \frac{q}{2} \right\rfloor\right) \\
=&\left(x\vec{t}G + \vec{e^*}\right) \cdot G^{-1}\left(\vec{0}\| \left\lfloor \frac{q}{2} \right\rfloor\right) \\
=&x\left\langle \vec{t}, \vec{0}\| \left\lfloor \frac{q}{2} \right\rfloor \right\rangle + \vec{e^*}G^{-1}\left(\vec{0}\| \left\lfloor \frac{q}{2} \right\rfloor\right) \\
=&x\left\lfloor \frac{q}{2} \right\rfloor + \hat{e} \text{ s.t. } |\hat{e}| \leq m\gamma
\end{aligned}$$

Consequently, if $m\gamma \leq \left\lfloor \frac{q}{4} \right\rfloor$, then a decryptor can distinguish $x = 1$ from $x = 0$ with probability 1. Now we show the first of our homomorphic operations, addition, which is simple enough:

Algorithm 7. $\mathsf{GSWAdd}(C_1, C_2)$**:**

1. output $C_1 + C_2$

If C_1 and C_2 are γ-noisy encryptions of two bits x_1, x_2, then the output C_{1+2} is a 2γ-noisy encryption of $x_1 + x_2$. This works because

$$\begin{aligned}
\vec{t}C_{1+2} &= \vec{t}(C_1 + C_2) \\
&= (x_1 + x_2) \cdot \vec{t}G + (\vec{e_1} + \vec{e_2})
\end{aligned}$$

Finally, we will show how to multiply. First, we must abuse our notation a bit more by defining G^{-1} over matrices (whereas untill now it has only been defined for vectors). Specifically, we define $G^{-1} : \mathbb{Z}_q^{n \times m} \mapsto \{0,1\}^{m \times m}$ to yield

$$G^{-1}(C) = G^{-1}(C_1)\|G^{-1}(C_2)\| \cdots \|G^{-1}(C_m)$$

where C_i is the i^{th} column of C expressed as a vector.

Algorithm 8. $\mathsf{GSWMul}(C_1, C_2)$**:**

1. output $C_1 G^{-1}(C_2)$

If C_1 and C_2 are γ-noisy encryptions of two bits x_1, x_2, then the output $C_{1\cdot 2}$ is a

$((m+1)\,\gamma)$-noisy encryption of $x_1 \cdot x_2$. This works because

$$\begin{aligned}
\vec{t}C_{1\cdot 2} &= \vec{t}C_1 G^{-1}(C_2) \\
&= (x_1 \cdot \vec{t}G + e_1)G^{-1}(C_2) \\
&= x_1 \cdot \vec{t}C_2 + e' \qquad \text{s.t. } e' = e_1 G^{-1}(C_2) \\
&= x_1 \left(x_2 \vec{t}G + e_2\right) + e' \\
&= x_1 \cdot x_2 \cdot \vec{t}G + x_1 \cdot e_2 + e' \\
&= x_1 \cdot x_2 \cdot \vec{t}G + e^* \qquad \text{s.t. } e^* = x_1 \cdot e_2 + e'
\end{aligned}$$

where $\|e^*\|_\infty = (m+1)\gamma$. Note that because the noise term $e^* = x_1 \cdot e_2 + e_1 G^{-1}(C_2)$ is dependent upon the two inputs in slightly different ways, the multiplication operation isn't precisely commutative. If the inputs are swapped, the output will decrypt to the same value, but the noise term may be (significantly) different. Finally, there is an easy trick to compute NAND of two bits in a single step, which is useful because NAND is complete for all computations:

Algorithm 9. $\mathsf{GSWNand}(C_1, C_2)$**:**

1. output $G - \mathsf{GSWMul}(C_1, C_2)$

Bear in mind that the three homomorphic operations we have described (addition/XOR, multiplication/AND, NAND) all return ciphertexts with higher noise than their input ciphertexts. As we have given no way to *reduce* the noise of a ciphertext, this means that the scheme is *leveled*, i.e. that it can compute funcitons only of a finite, pre-determined depth. Since NAND is complete for computation, this will be easiest to reason about in the context of NAND-logic. Remember that the decryption algorithm is perfectly correct only for γ-noisy ciphertexts such that $m\gamma \leq \left\lfloor \frac{q}{4} \right\rfloor$. For a fresh encryption, $\gamma \leq m\beta$, and for a NAND gate with two γ-noisy inputs, the output ciphertext is $((m+1)\,\gamma)$-noisy. Consequently, the noise level of an entire NAND-circuit is given by $\gamma \leq (m+1)^d\, m\beta$, where d is the depth of the circuit, and to fulfill the conditions for decryption, we need

$$(m+1)^d\, m^2 \beta \leq \left\lfloor \frac{q}{4} \right\rfloor$$

which yields circuits of a maximum depth d such that

$$d \leq \frac{\log\left(\frac{\left\lfloor \frac{q}{4} \right\rfloor}{m^2\beta}\right)}{\log(m+1)}$$

8 The Security of GSW

The security of the GSW scheme can be shown via the hardness of DLWE and the Left-over Hash Lemma, and follows an argument quite similar to that of the security of Regev

Encryption. In short, we wish to show that

$$\left\{\left(\bar{A}, \bar{A}R\right) : (\bar{A}, \mathsf{sk}) \leftarrow \mathsf{GSWGen}(n, m, \chi), R \leftarrow \{0,1\}^{m\times m}\right\} \\ \stackrel{c}{\equiv} \left\{(U_1, U_2) : U_1 \leftarrow \mathbb{Z}_q^{n\times m}, U_2 \leftarrow \mathbb{Z}_q^{n\times m}\right\}$$

If this is true, then it follows directly that

$$\left\{\left(\bar{A}, \bar{A}R + x \cdot G\right) : (\bar{A}, \mathsf{sk}) \leftarrow \mathsf{GSWGen}(n, m, \chi), R \leftarrow \{0,1\}^{m\times m}, x \in \{0,1\}\right\} \\ \stackrel{c}{\equiv} \left\{(U_1, U_2) : U_1 \leftarrow \mathbb{Z}_q^{n\times m}, U_2 \leftarrow \mathbb{Z}_q^{n\times m}\right\}$$

Note that $\bar{A}$ is exactly the form of the public key in the GSW scheme, and $\bar{A}R + x \cdot G$ is exactly the form of an encryption of the bit x. This being the case, if the first condition is true, then we will have proven indistinguishability of ciphertexts under chosen plaintext attack, as specified in Definition 3. Thus, our security theorem:

Theorem 2.

$$\left\{\left(\bar{A}, \bar{A}R\right) : (\bar{A}, \mathsf{sk}) \leftarrow \mathsf{GSWGen}(n, m, \chi), R \leftarrow \{0,1\}^{m\times m}\right\} \\ \stackrel{c}{\equiv} \left\{(U_1, U_2) : U_1 \leftarrow \mathbb{Z}_q^{n\times m}, U_2 \leftarrow \mathbb{Z}_q^{n\times m}\right\}$$

Proof. The proof will proceed via a series of hybrids:

H_0 The left hand side of the theorem statement. That is,

$$\left\{\left(\bar{A}, \bar{A}R\right) : (\bar{A}, \mathsf{sk}) \leftarrow \mathsf{GSWGen}(n, m, \chi), R \leftarrow \{0,1\}^{m\times m}\right\}$$

H_1 $\bar{A}$ is replaced by $U_1 \leftarrow \mathbb{Z}_q^{n\times m}$, yielding the tuple

$$\left\{(U_1, U_1R) : U_1 \leftarrow \mathbb{Z}_q^{n\times m}, R \leftarrow \{0,1\}^{m\times m}\right\}$$

Note that $\bar{A}$ comprises the A and $\vec{b}$ components of the public key for Regev encryption, where $A \leftarrow \mathbb{Z}_q^{n\times(m-1)}$ and $\vec{b} := \vec{s}A + \vec{e}$. Thus $H_1 \stackrel{c}{\equiv} H_0$ by the assumed hardness of the Decisional LWE problem.

H_2 U_1R is replaced by $U_2 \leftarrow \mathbb{Z}_q^{n\times m}$, yielding the right hand side of the theorem statement:

$$\left\{(U_1, U_2) : U_1 \leftarrow \mathbb{Z}_q^{n\times m}, U_2 \leftarrow \mathbb{Z}_q^{n\times m}\right\}$$

If we consider U_1R as a hash function:

$$H_{\mathbb{Z}_q^{n\times m}} : \{0,1\}^{m\times m} \mapsto \mathbb{Z}_q^{n\times m} \quad \text{s.t.} \quad H_{U_1}(R) = U_1R$$

we can see that it is a compressing hash function if $m > n \log q$, and more importantly that it is a universal hash function:

$$\Pr_{U_q, R}\left[H_{U_1}(R) = H_{U_1}(R') | R \neq R'\right] = \frac{1}{q^{nm}}$$

Thus by applying the Leftover Hash Lemma, we find that

$$\left\{H_{U_1}(R) : U_1 \leftarrow \mathbb{Z}_q^{n\times m}, R \leftarrow \mathbb{Z}_q^{m\times m}\right\} \stackrel{s}{\equiv} \left\{U_2 \leftarrow \mathbb{Z}_q^{n\times m}\right\}$$

and consequently $H_1 \stackrel{s}{\equiv} H_2$, which implies $H_0 \stackrel{c}{\equiv} H_2$ □