On the Implementation of Modified Fuzzy Vault for Biometric Encryption

Xinmiao Zhang
Case Western Reserve University
xinmiao.zhang@case.edu

Richard Shi and James Ritcey
University of Washington
{cjshi, jar7}@uw.edu

***Abstract*—Biometrics, such as irises and fingerprints, enable secure and non-repudiable authentication. Fuzzy vault is a scheme that can monolithically bind secret to biometric templates. Moreover, the modified fuzzy vault (MFV) leads to less entropy loss and requires less memory for storing the sketches. This paper proposes a novel low-complexity scheme to compute the monic polynomial for the sketch during the enrollment process of the MFV. An innovative interpolation method is also developed to reduce the computation complexity and latency of the verification process. Efficient hardware implementation architectures are developed in this paper for the proposed schemes and their complexities are analyzed in detail.**

I. Introduction

Compared to traditional passwords, biometrics, such as fingerprints and irises, have the advantage that they are unique to individuals, and can not be forgotten or lost. Hence, biometrics enable higher level of security. On the other hand, the biometric templates acquired from the same user can be slightly different, and calls for error-correction. In addition, since biometrics are not replaceable, they should not be stored directly in the database in case it gets compromised. To address these issues, biometric encryption schemes have been developed [1]–[4] to marry encryption with error-correction. In these schemes, secret information are bound with biometric templates, and neither the secret nor the enrolled biometric template can be derived from the sketches stored in the database, unless another very similar biometric template is provided.

Due to the natural fuzziness of biometrics, the template acquired for verification may have deleted or added symbols compared to the enrolled template of the same user. The fuzzy commitment [1] and fuzzy syndrome hashing [4] schemes employ linear block codes, and the decoding is done based on the parity check matrix. All the symbols from a template are arranged in a certain order to be decoded. In case there are deletion or addition of symbols, the location of many symbols will be shifted. This causes decoding failure and accordingly high false rejection rate. On the contrary, the fuzzy vault scheme [2] adopts interpolation-based decoding of Reed-Solomon (RS) codes, and is indifferent to addition, erasure, or the order of the interpolation points. The information stored for a user and accessible during the verification process is called the sketch. The sketch in the fuzzy vault scheme includes real points related to the biometric template (usually less than 40 for fingerprints) and a much larger number of random chaff points (usually hundreds for fingerprints) to hide the real points. Comparatively, the sketch in the modified fuzzy vault (MFV) [3] consists of the coefficients of a monic polynomial whose degree equals the number of real points. As a result, the sketch in the MFV requires much smaller memory for storage, and reveals less information about the user template.

In this paper, novel complexity-reduction schemes are developed for implementing the MFV scheme, and fingerprint biometric is mainly considered. In the enrollment process, a monic polynomial that passes a set of given points needs to be derived for the sketch. Inspired by the Kötter's interpolation algorithm [5] for constructing polynomials with minimum weighted degree, a modified interpolation process is developed to efficiently construct such a monic polynomial without solving linear equations. The verification process computes a polynomial with minimum weighted degree that passes a set of points derived by evaluating the monic polynomial over the symbols in the query template. Although this can be done by the Kötter's algorithm, the polynomial evaluation needs to be carried out first. Alternatively, this paper proposes to use the monic polynomial directly in the construction of an initial basis, which is then converted to compute the polynomial with minimum weighted degree using the Lee-O'Sullivan (LO) [6] interpolation process. Our approach avoids not only the polynomial evaluation, but also the complicated computations for initial basis construction. Efficient VLSI architectures are also developed in this paper for implementing the proposed schemes, and the complexities are analyzed in detail.

The structure of this paper is as follows. Section II introduces the MFV and prior interpolation algorithms. Section III presents the modified interpolation for monic polynomial construction and modified initial basis construction for the verification process. Hardware implementation architectures are developed, and their complexities are analyzed in Section IV. Conclusions follow in Section V.

II. Modified fuzzy vault and interpolation

The fuzzy vault schemes are built upon (n, k) RS codes over $GF(2^m)$ with evaluation map encoding and algebraic interpolation-based decoding. k equals the number of secret

This material is based upon the work supported by NSF grants 0846331 and 0835782.

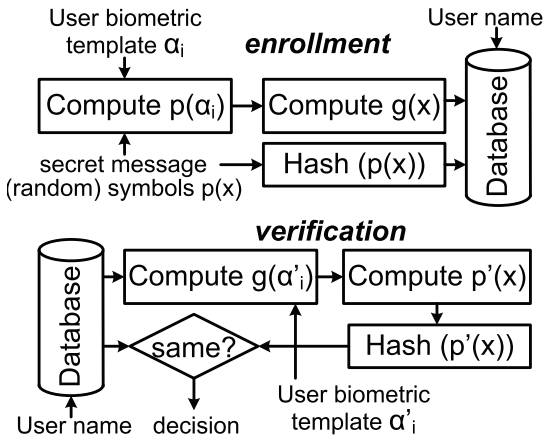

Fig. 1. The modified fuzzy vault scheme

message symbols. In authentication-only applications, k random symbols can be used. The order of the finite field is decided by the resolution required for biometric images. For example, for a fingerprint image, 8-bit resolution is usually needed for each coordinate, and the two coordinates are put together as a symbol over $GF(2^{16})$. The template of a fingerprint consists of the locations of the minutiae, which are the ridge endings or bifurcations in the fingerprint image. Typically, there are 20∼40 minutiae in a fingerprint template, and the number of minutiae decides n, the codeword length of the adopted RS code. Between two templates acquired from the same user, the number of different minutiae can be more than 25%. An (n, k) RS code can correct as many as $\lfloor (n-k)/2 \rfloor$ errors using traditional hard-decision decoding. Hence, to achieve reasonable false rejection rate, k can not be a large number in the fuzzy vault schemes for fingerprints.

The MFV scheme is shown in Fig. 1. The k secret message symbols can be considered as the coefficients of a degree $k-1$ message polynomial $p(x)$. During the enrollment, given the n symbols from the user biometric, $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, $\beta_i = p(\alpha_i)$ $(0 \le i < n)$ are first computed. This can be considered as the evaluation map encoding of RS codes. Then a degree n monic polynomial $g(x)$ that passes each (α_i, β_i) is computed. In addition, a hash function is applied to $p(x)$. The hash value and the coefficients of $g(x)$ are stored as the sketch.

During the verification process, the user provides fingerprint template with symbols $\alpha'_0, \alpha'_1, \ldots, \alpha'_{n'-1}$. Note that n and n' can be different, even if the template is from the authorized user. $\beta' = g(\alpha'_i)$ are computed, and then interpolation is carried out to find a bivariate polynomial, $Q(x, y)$, of minimum $(1, k-1)$ weighted degree that passes each point (α'_i, β'_i) $(0 \le i < n')$. If the number of errors is within correctable range, then it is guaranteed that $Q(x, y)$ has a factor $y - p'(x)$ with $\deg(p'(x)) < k$. This is the same process as the algebraic interpolation-based decoding for RS codes proposed in [7]. If the hash value of $p'(x)$ matches the stored hash value, access is granted or the computed $p'(x)$ is verified the same as the secret message $p(x)$. In the case of no matching, no additional information is told by the verification.

The implementation of hash functions has been addressed in many papers. This work focuses on the implementation of the other computations involved in the enrollment and verification process of the MFV scheme. The following definitions are necessary to understand the materials in this paper.

Definition 1: The (w_x, w_y)-weighted degree of a monomial $x^a y^b$ is $aw_x + bw_y$. For a bivariate polynomial $Q(x, y) = \sum\sum q_{a,b} x^a y^b$, its (w_x, w_y)-weighted degree is the maximum of $aw_x + bw_y$ such that $q_{a,b} \neq 0$.

Definition 2: The monomial with the highest lexicographic order in a polynomial is called the leading term, and is denoted by $lt(\cdot)$. The corresponding coefficient is the leading coefficient, and is represented by $lc(\cdot)$.

Definition 3: A set of nonzero polynomials, $Q^{(0)}(x, y)$, $Q^{(1)}(x, y), \ldots, Q^{(t)}(x, y)$ form a Gröbner basis of a module $\mathcal{M}$, if for each polynomial, $F(x, y)$, in the module $\mathcal{M}$, there exists $l \in \{0, 1, 2, ..., t\}$ such that $lt(Q^{(l)}(x, y))$ divides $lt(F(x, y))$. A monomial $x^a y^b$ is said to divide $x^{a'} y^{b'}$ when $a \le a'$ and $b = b'$.

From Definition 3, the polynomial of minimum weighted degree in a module must be the polynomial of minimum weighted degree in its Gröbner basis. Hence, the polynomial of $(1, k-1)$ weighted degree, $Q(x, y)$, needed in the verification process that passes each interpolation point can be found by constructing a Gröbner basis of the polynomials passing the same points. Such a task can be accomplished efficiently by using the Kötter's or LO interpolation algorithms. The number of monomials in $Q(x, y)$ should be larger than the number of interpolation constraints, C, in order to guarantee a nonzero solution. The monomials can be arranged according to increasing $(1, k-1)$ weighted lexicographic order. Then the maximum y-degree, t, of the interpolation solution, and hence the number of polynomials in the Gröbner basis, can be decided from the maximum y-degree of the first $C+1$ monomials [8].

To find a polynomial of minimum $(1, k-1)$ weighted degree that passes each point (α_i, β_i) once, the Kötter's interpolation algorithm can be simplified as in Algorithm A.

Algorithm A: Kötter's Interpolation Algorithm

Initialize:

$Q^{(0)}(x, y) = 1$, $Q^{(1)}(x, y) = y$, ..., $Q^{(t)}(x, y) = y^t$

$Wdeg_0 = 0$, $Wdeg_1 = k-1$, ..., $Wdeg_t = (k-1)t$

Start:

for each (α_i, β_i)

A1: compute $Q^{(l)}(\alpha_i, \beta_i)$ $(0 \le l \le t)$

$minl \leftarrow \arg\min_l \{Wdeg_l | Q^{(l)}(\alpha_i, \beta_i) \neq 0, 0 \le l \le t\}$

for $l = 0$ to t, $l \neq minl$

A2: $Q^{(l)}(x, y) \Leftarrow Q^{(minl)}(\alpha_i, \beta_i) Q^{(l)}(x, y) + Q^{(l)}(\alpha_i, \beta_i) Q^{(minl)}(x, y)$

A3: $Q^{(minl)}(x, y) \Leftarrow Q^{(minl)}(x, y)(x + \alpha_i)$

$Wdeg_{minl} \Leftarrow Wdeg_{minl} + 1$

Output: $Q^{(\varphi)}(x, y)$, $\varphi = \arg min\{Wdeg_l | 0 \le l \le t\}$

The updating in the A2 and A3 steps of the Kötter's algorithm forces each polynomial, $Q^{(l)}(x, y)$, to pass (α_i, β_i) with minimum increase in the weighted degree. In addition, the

polynomials still pass the points covered in previous iterations after the updating. At the end of each iteration, the $t+1$ polynomials form a Gröbner basis of the polynomials that pass all points interpolated so far. Hence, the desired interpolation output polynomial can be found from the Gröbner basis constructed at the end of the last iteration.

Different from the Kötter's algorithm, the LO interpolation starts with a basis of $t+1$ polynomials that satisfy all interpolation constraints. Then it is converted to a Gröbner basis, which can be defined equivalently as a basis whose polynomials have distinct y-degree in the leading terms [6]. Assume that $Q^{(l)}(x,y) = q_0^{(l)}(x) + q_1^{(l)}(x)y + \cdots + q_t^{(l)}(x)y^t$. The LO algorithm can be carried out according to Algorithm B. In this algorithm, $(1, k-1)$ weighted lexicographic order is adopted in deciding the leading term.

Algorithm B: Lee-O'Sullivan Interpolation Algorithm

Input: A basis $\{Q^{(l)}(x,y)\}$ $(0 \le l \le t)$ that satisfies all interpolation constraints

Initialize: $r = 0$

Start:

B1: $r = r+1$; proceed if $r \le t$; otherwise, go to *Output*

B2: find $s = y$-degree of $lt(Q^{(r)}(x,y))$
if $s == r$, then go to step B1

B3: $d = deg(q_s^{(r)}(x)) - deg(q_s^{(s)}(x))$

B4: if $d \ge 0$, then
$Q^{(r)}(x,y) = lc(q_s^{(s)}(x))Q^{(r)}(x,y) + x^d lc(q_s^{(r)}(x))Q^{(s)}(x,y)$
else
$tmp(x,y) = Q^{(r)}(x,y)$
$Q^{(r)}(x,y) = x^{-d} lc(q_s^{(s)}(x))Q^{(r)}(x,y) + lc(q_s^{(r)}(x))Q^{(s)}(x,y)$
$Q^{(s)}(x,y) = tmp(x,y)$
goto step B2

Output: $Q^{(l)}(x,y)$ with the minimum weighted degree

It was proposed in [6] to construct the initial basis as

$$Q^{(l)}(x,y) = \begin{cases} \prod(x+\alpha_i), & l = 0 \\ y - h(x), & l = 1 \\ y^{l-1}(y-h(x)), & 2 \le l \le t \end{cases} \quad (1)$$

where $h(x)$ is a polynomial passing each (α_i, β_i) derived by using the Lagrange interpolation. Accordingly, each $Q^{(l)}(x,y)$ in (1) passes all interpolation points. The polynomials are also linearly independent and have different y-degree. Hence, they form a basis of the polynomials that pass all points. Starting from such a basis, the updating in Step B4 of Algorithm B cancels out the leading term of $Q^{(r)}(x,y)$ iteratively, until the leading term has y-degree as r. In addition, this updating does not change the y-degree of the leading term in $Q^{(s)}(x,y)$, and the $t+1$ polynomials still form a basis of the same module after the updating. After the iterations are carried out for $r = 1, 2, \ldots, t$, the $t+1$ polynomials in the basis have distinct y-degree in the leading terms, and hence form a Gröbner basis.

III. Modified interpolation and basis construction

This section proposes a modified interpolation algorithm for calculating the monic polynomial, $g(x)$, in the enrollment process, and a modified basis construction for the LO interpolation that can be used in the verification process of the MFV.

A. Modified interpolation

The monic polynomial, $g(x) = x^n + \sum_{j=0}^{n-1} g_j x^j$, needs to pass each of the n points (α_i, β_i) $(0 \le i < n)$, where α_i are distinct. It was suggested in [3] to find the n coefficients, g_j, by solving n linear equations $g(\alpha_i) = \beta_i$. This process has complexity $O(n^3)$, and is not hardware friendly. On the other hand, the Kötter's and LO interpolation algorithms generate polynomials of minimum weighted degree that pass each point through constructing Gröbner bases. No matter what weighted degree is adopted, these algorithms can not guarantee to output a monic polynomial of degree n that passes each of the n points. Nevertheless, the idea from the Kötter's algorithm on forcing polynomials to pass a point can be borrowed to construct a monic polynomial iteratively. Our proposed modified interpolation is listed in Algorithm C.

Algorithm C: Modified Interpolation for Monic Polynomial Construction

Initialize: $Q^{(0)}(x,y) = 1$, $Q^{(1)}(x,y) = y + x^n$

Start:

for $i = 0$ to $n-1$

C1: compute $Q^{(l)}(\alpha_i, \beta_i)$ $(l = 0, 1)$

C2: $Q^{(1)}(x,y) \Leftarrow Q^{(1)}(\alpha_i,\beta_i)Q^{(0)}(x,y) + Q^{(0)}(\alpha_i,\beta_i)Q^{(1)}(x,y)$

C3: $Q^{(0)}(x,y) \Leftarrow Q^{(0)}(x,y)(x+\alpha_i)$

Output: $Q^{(1)}(x,y)$

In Algorithm C, only two polynomials are involved. To show the connections with the Kötter's algorithm, bivariate notations are used. However, $Q^{(0)}(x,y)$ starts from 1 and is iteratively multiplied by $(x+\alpha_i)$. Hence it is actually a univariate polynomial with $q_1^{(0)}(x) = 0$. At the beginning of iteration i, $Q^{(0)}(x,y) = \prod_{j=0}^{i-1}(x+\alpha_j)$. Since α_i are all distinct for fingerprint applications, $Q^{(0)}(\alpha_i,\beta_i) = \prod_{j=0}^{i-1}(\alpha_i+\alpha_j) \ne 0$. Accordingly, through the linear combination updating in Step C2 of Algorithm C, $Q^{(1)}(x,y)$ is forced to pass (α_i,β_i) in a similar way as that in the Kötter's Algorithm. In addition, $Q^{(1)}(x,y)$ still passes (α_j,β_j) for $j < i$ after the updating, since both $Q^{(1)}(x,y)$ and $Q^{(0)}(x,y)$ pass those points at the beginning of iteration i. The degree of $Q^{(0)}(x,y)$ increases from 0 at the beginning of iteration 0 to $n-1$ at the beginning of iteration $n-1$. Hence, the linear combination in Step C2 does not involve any addition on monomials y and x^n in $Q^{(1)}(x,y)$, although they can be multiplied by the same finite field element. Therefore, at the end of Algorithm C, $Q^{(1)}(x,y)$ is in the format of $ay + ax^n + \sum_{j=0}^{n-1} g'_j x^j$

($a \in GF(2^m)$). Multiplying the inverse factor a^{-1} to each coefficient, a monic polynomial that passes each point can be derived. $Q^{(1)}(x,y)$ can be also updated as $Q^{(1)}(x,y) + (Q^{(1)}(\alpha_i,\beta_i)/Q^{(0)}(\alpha_i,\beta_i)) \times Q^{(0)}(x,y)$ in Step C2 to avoid the inverse factor multiplications at the end. It also reduces the number of multiplications needed for each iteration. However, calculating the inverse of finite field elements in each iteration slows down the interpolation process.

Although the computations in Algorithm C look like those in Algorithm A with $(1, J)$ weighted degree and $J \geq n$, the ideas behind the polynomial initialization and updating rule in Algorithm C are different. There are n interpolation points in total, and passing each of them requires multiplying a factor $(x+\alpha_i)$ to one of the interpolation polynomials if approaches similar to that in Algorithm A are adopted. Therefore, the x-degree of an interpolation polynomial will not exceed n if it is initially zero. On the other hand, the polynomial of interest is in the format of $y + x^n + \ldots$. The y and x^n terms will never get affected during the interpolation process if this polynomial is never multiplied by any $(x + \alpha_i)$ factor, and always forced to pass (α_i, β_i) through linearly combining with another polynomial that has lower degree and nonzero evaluation value over each (α_i, β_i). All these problems can be solved by multiplying the n $(x + \alpha_i)$ factors iteratively to the same polynomial whose initial x-degree is zero, and using this polynomial to linearly update the other polynomial that is initialized as $y + x^n$.

To construct the desired monic polynomial that passes each of the n points, n iterations need to be run in Algorithm C. Moreover, the maximum x-degree of the two involved polynomials is n. Hence, the complexity of Algorithm C is $O(n^2)$. Compared to solving a set of n linear equations as suggested by [3], the proposed algorithm is much simpler. Moreover, it can be implemented by efficient and regular architectures as will be shown in the next section.

B. Modified basis construction

During the verification process of the MFV scheme, the original procedure in [3] is to first evaluate the monic polynomial $g(x)$ over each symbol, α'_i, of the query template to derive a set of n' points $(\alpha'_i, \beta'_i = g(\alpha'_i))$ $(0 \leq i < n')$. Then $p'(x)$ can be recovered as a factor of a bivariate polynomial $Q(x, y)$, which has minimum $(1, k-1)$ weighted degree and passes each point. Given the set of points, either the Kötter's or the LO interpolation algorithm can compute such a bivariate polynomial. The re-encoding and coordinate transformation [9], [10] is a popular technique that can reduce the number of points to be interpolated from n' to $n' - k$ at the cost of two systematic encoders and a Berlekamp-Massey decoder [11]. Considering the overhead, and that k is much smaller than n' in biometric encryption to tolerate the large intra-user variation, this technique is not adopted in our design.

$g(x)$ already passes each point (α'_i, β'_i), and the information of all points is already incorporated in this polynomial. Although the polynomial to be computed in the verification process, $p'(x)$, has lower degree, it passes the same points. Intuitively, $g(x)$ should contribute directly to the computation of $p'(x)$. The Kötter's algorithm iteratively builds a Gröbner basis for polynomials passing one additional point at a time. Since $g(x)$ already passes all the points, it is very difficult to make use of $g(x)$ in the Kötter's interpolation.

Compared to the Kötter's interpolation, the LO algorithm has simpler computations in each iteration. Moreover, it starts from an initial basis in the format of (1) that passes all the points. Then the initial basis is converted to a Gröbner basis. In order for the second and third polynomials in (1) to pass each point, $h(x)$ needs to pass each point. Moreover, $h(x)$ should not have any common factor with $\prod_{i=0}^{n'-1}(x + \alpha'_i)$ in the first equation of (1). Otherwise, (1) is not a proper basis, and any polynomial, $F(x, y) = f_0(x) + f_1(x)y + \ldots$ in the corresponding module is subject to the constraint that it has the same common factor in $f_0(x)$. Hence, given the set of points to be passed, the Lagrange formula is adopted to calculate $h(x)$ in [6]. However, the monic polynomial $g(x)$ computed from the modified interpolation satisfies the same criteria, and can replace $h(x)$. Hence, the LO interpolation can start from the basis

$$Q^{(l)}(x,y) = \begin{cases} \prod_{i=0}^{n'-1}(x+\alpha'_i), & l = 0 \\ y - g(x), & l = 1 \\ y^{l-1}(y - g(x)), & 2 \leq l \leq t \end{cases} \quad (2)$$

Since $g(x)$ is stored in the database, it can be used directly. As a result, evaluating $g(x)$ over each α'_i is not necessary. Moreover, using $g(x)$ directly to construct the initial basis avoids the complicated Lagrange interpolation for computing $h(x)$. In the LO interpolation shown in Algorithm B, the number of iterations needed to make the y-degree of $lt(Q^{(l)}(x,y))$ equal l is decided by the differences of the degrees of $q_0^{(l)}(x), q_1^{(l)}(x), \ldots, q_l^{(l)}(x)$ at initialization. $\deg(h(x)) = n'-1$ and $\deg(g(x)) = n$. Since n and n' are the numbers of symbols in the biometric templates from different acquisitions, both $n' > n$ and $n' < n$ can happen. Therefore, using the modified initialization does not necessarily increase the complexity of the LO interpolation process.

IV. VLSI Architectures

In this section, efficient VLSI architectures are presented for implementing the proposed schemes. Biometric encryption systems do not need very high speed. Delay of less than a millisecond is short enough. The goal of our design is to achieve this speed with minimized silicon area. Fingerprint encryption is considered in our design, and $GF(2^{16})$ is adopted for hardware complexity analysis. k is set to 9, so that 144-bit information can be hidden in the biometric. In the case that there are $n = 21$ symbols in the template, six errors, which are 6/21=28% of the codeword length, can be corrected.

A. VLSI architectures for enrollment

By applying the Horner's rule, $\beta_i = p(\alpha_i)$ can be computed in $\deg(p(x)) + 1 = k$ clock cycles using the univariate polynomial evaluation (UPE) architecture shown in Fig. 2. α_i and β_i are stored into the memory after the computation.

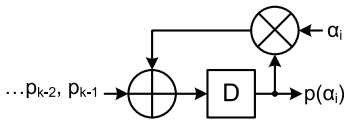

Fig. 2. Architecture for univariate polynomial evaluation (UPE)

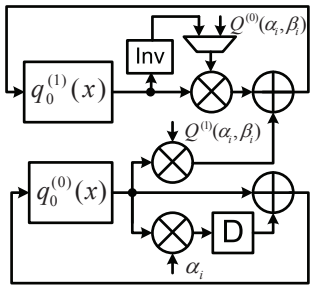

Fig. 3. Polynomial updating (PU) architecture for modified interpolation

Each iteration of the proposed modified interpolation for constructing $g(x)$ consists of two parts: polynomial evaluation in Step C1 of Algorithm C, and polynomial updating in Step C2 and C3. For the two involved polynomials, $Q^{(l)}(\alpha_i, \beta_i) = q_1^{(l)}(\alpha_i)\beta_i + q_0^{(l)}(\alpha_i)$. As mentioned previously, $Q^{(0)}(x, y)$ is actually a univariate polynomial with $q_1^{(0)}(x) = 0$, Hence, $Q^{(0)}(\alpha_i, \beta_i)$ can be computed by sending the coefficients of $q_0^{(0)}(x)$ to the UPE architecture. In addition, the evaluations of $q_0^{(0)}(x)$ and $p(x)$ can share the same UPE unit since they are not done at the same time. $q_0^{(1)}(\alpha_i)$ can be computed by another copy of the UPE unit. Moreover, $q_1^{(1)}(x)$ only has a nonzero constant term $q_{1,0}^{(1)}$. Therefore, by adding multiplexors to the UPE unit for $q_0^{(1)}(\alpha_i)$ computation, $q_{1,0}^{(1)}\beta_i$ can be computed and added to $q_0^{(1)}(\alpha_i)$ using one more clock cycle. In total, two copies of the UPE units are needed to evaluate $p(x)$ and $Q^{(l)}(x, y)$ $(l = 0, 1)$, although one of the copies needs to have two multiplexors added. Once $Q^{(l)}(\alpha_i, \beta_i)$ are computed, they are sent to the polynomial updating (PU) architecture in Fig. 3 to carry out the C2 and C3 steps for polynomial updating.

In Fig. 3, the coefficient $q_{1,0}^{(1)}$ is stored in the same memory block as those of $q_0^{(1)}(x)$, and the memory cells are initialized while $p(\alpha_i)$ are being computed. One coefficient is read from each memory and updated at a time, starting from the most significant coefficient. The two multipliers and adder on the top implement the C2 step, and the multiplier-register-adder units on the bottom carry out the C3 step. The updated coefficients are written back to the memory. Since both the PU and UPE architectures process the polynomial coefficients serially with the most significant ones first, the updated coefficients from the PU unit can be sent to the UPE unit right after to start the evaluation value computation for the next iteration. After the last iteration, the inverse of the coefficient of y is multiplied to each coefficient stored in the $q_0^{(1)}(x)$ memory to derive the monic $g(x)$. For security reason, α_i and β_i should be cleared from the memory after they have used in the polynomial updating, and $q_0^{(0)}(x)$ should be erased during the last iteration of the modified interpolation.

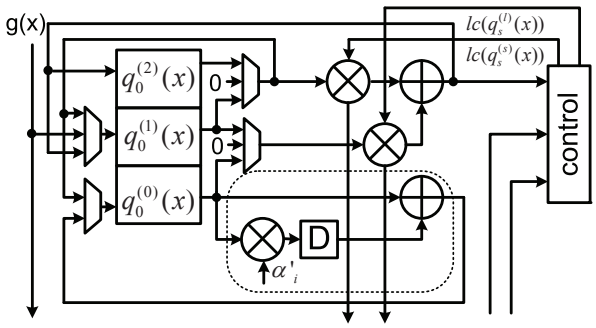

Fig. 4. Interpolation architecture for verification process

B. VLSI architectures for verification

The number of polynomials involved in the interpolation of the verification process is affected by k and n'. n' varies with the query template. In the case of $k = 9$, it can be computed that $t = 1$ if $n' < 24$ and $t = 2$ if $24 \leq n' < 48$. Hence, up to three polynomials can be involved in the interpolation for fingerprint encryption, and our architecture is designed to handel this case.

Both the modified basis construction and the LO algorithm can be implemented by the architecture in Fig. 4. This architecture also processes the coefficients in a polynomial serially. The units in the dashed block multiply one $(x + \alpha'_i)$ at a time, and compute $\prod_{i=0}^{n'-1}(x + \alpha'_i)$ iteratively for basis initialization according to (2). Since $\deg(\prod_{j=0}^{i-1}(x + \alpha'_j)) = i$, the multiplication of $(x + \alpha'_i)$ takes $(i + 2)$ clock cycles. While these multiplications are being carried out, the memory for $Q^{(1)}(x, y)$ and $Q^{(2)}(x, y)$ are initialized using $g(x)$. The control functions in Step B1∼B3 steps of Algorithm B and that for switching the polynomials in Step B4 are implemented by the 'control' unit and multiplexors in Fig. 4. The linear combination updating of the polynomial in Step B4 is implemented by the two multipliers and one adder in the top middle. For the purpose of conciseness, this figure only shows the units for processing $q_0^{(l)}(x)$ $(l = 0, 1, 2)$. There are another two sets of units for processing $q_1^{(l)}(x)$ and $q_2^{(l)}(x)$.

The memory blocks for storing $q_j^{(l)}(x)$ $(0 \leq l, j \leq 2)$ have separate address generators and enable signals. Only the memories for $q_j^{(r)}(x)$ and $q_j^{(s)}(x)$ with $j \leq r$ are enabled in the iterations for converting $Q^{(r)}(x, y)$. The purpose of the multiplication by $x^{|d|}$ in Step B4 is to make sure that the leading terms of $Q^{(r)}(x, y)$ and $Q^{(s)}(x, y)$ are aligned. It can be achieved by using proper addresses for memory access. Nevertheless, for the linear combination, zeros need to be padded after the least significant coefficients of the polynomial that has been multiplied by $x^{|d|}$. The '0' inputs to the two multiplexors in the middle of Fig. 4 are used for this purpose. The linear combination of the polynomials may cancel out not only the leading term of $q_s^{(r)}(x)$, but also other following terms. Hence, zero testing needs to be done in the control block to tell the real degree of $q_s^{(r)}(x)$ after the linear combination. The leading coefficient and degree of each $q_j^{(l)}(x)$ $(0 \leq l, j \leq 2)$ are stored in the control block. Their initial values can be easily derived from (2). They can

TABLE I
HARDWARE COMPLEXITY OF FINGERPRINT BIOMETRIC ENCRYPTION WITH $n = n' = 30$ AND $k = 9$

	GF Mult.	GF Add.	GF Inv.	2:1 m-bit Mux.	m-bit Regis.	RAM (bits)	Latency (clks)
$p(x)$ Eval. & Mod. Interp.	5	4	1	3	3	$124m$	270 +1024
Mod. Basis Constr. & LO Interp.	7	4	0	21	1	$216m$	495 +1743

be only replaced by that of another polynomial or updated as a result of the linear combination according to step B4. Hence, the control block is able to derive the most updated leading coefficient and degree of each $q_j^{(l)}(x)$ through tracking the polynomial updating and the coefficients in the linearly combined polynomial. Then proper coefficients are chosen as $lc(q_s^{(r)}(x))$ and $lc(q_s^{(s)}(x))$.

To find the factor of the interpolation output in the format of $y - p'(x)$, the Roth-Ruckenstein algorithm [12] can be employed. In the case of $t \leq 2$, an efficient architecture for implementing this algorithm can be found in [9], and is not repeated in this paper.

C. Hardware complexity analysis

Table I summarizes the hardware complexity of the proposed architectures for fingerprint biometric encryption. The numbers of functional units, such as multipliers and adders over $GF(2^m)$, can be directly counted from the proposed architectures. During the enrollment, the n points (α_i, β_i) need to be stored temporarily, and each of the $q^{(0)}(x)$ and $q^{(1)}(x)$ memory blocks in Fig. 3 has $n + 2$ coefficients. Using one UPE unit, the β_i for n points can be calculated in kn clock cycles. In the modified interpolation, updating or evaluating the polynomials take $n+2$ clock cycles each time. Considering that the polynomial updating in the last iteration can not be overlapped with any evaluation value computation, and the inverse factor needs to be multiplied at the end, the modified interpolation requires around $(n + 2)^2$ clock cycles.

In the interpolation for verification, computing the initial polynomial $\prod_{i=0}^{n'-1}(x+\alpha_i')$ takes $\sum_{i=0}^{n'-1}(i+2) = n'(n'+3)/2$ clock cycles. For the LO process, the latency and the size of each memory block in Fig. 4 are dependent on the x-degrees of the univariate polynomials $q_j^{(l)}(x)$ $(0 \leq l, j \leq 2)$. The linear combination in Step B4 can cancel out multiple coefficients at a time, and the x-degree also changes when the polynomials are switched. These issues make it very difficult to express the latency and memory size of the LO process in terms of the code parameters, n' and k. Hence, the latency and memory size are provided for an example case with $n = n' = 30$ and $k = 9$ in Table I. In this example, the template for verification has ten different symbols from that acquired in the enrollment. When the number of errors is smaller, the interpolation process has shorter latency. For example, when there are only two different symbols, the latency of the LO process can drop from 1743 to 1169 clock cycles.

Finite field additions are bit-wise XOR operations. $GF(2^{16})$ can be constructed from irreducible polynomials of weight five. Using standard basis representation of finite field elements, a $GF(2^{16})$ multiplier can be implemented by around 413 XOR gates with 16 gates in the critical path using the architecture in [13]. Accordingly, it can be observed from Fig. 2∼4 that the critical paths of the proposed architectures have at most 20 gates. Using CMOS technology, a clock period of 20ns can be easily achieved. As a result, the latency of the proposed architectures are less than 0.05ms. Due to the high order of the involved finite field, implementing the inversion using look-up tables would cause large area. Alternatively, composite field arithmetic can be adopted to decompose the computations into lower order fields. In this case, a $GF(2^{16})$ inverter can be implemented by less than 1K XOR gates instead of a 1024K-bit look-up table. To match the critical path of the rest architecture, the composite field inverter can be pipelined.

V. CONCLUSIONS

This paper proposed novel modified interpolation and basis construction schemes for MFV biometric encryption. Through making use of the properties of biometric templates and exiting interpolation algorithms, the proposed schemes bypass unnecessary computations and lead to significant complexity reduction. In addition, efficient implementation architectures were proposed and hardware complexity was analyzed. The MFV scheme shares the same theory as algebraic interpolation-based RS decoding. Nevertheless, it involves different types of codes and settings, and renders previous decoder designs inefficient or unusable. Developing low-cost and secure biometric encryption architectures will be the focus of our future research.

REFERENCES

[1] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," *Proc. 6th ACM Conf. on Computer and Commu. Security*, pp. 28-36, 1999.

[2] A. Juels and M. Sudan, "A fuzzy vault scheme," *Proc. IEEE Intl. Symp. Info. Theory*, 2004.

[3] Y. Dodis, L. Reyzin and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data," *Proc. Advances in cryptology-Eurocrypt*, pp. 523-540, 2004.

[4] E. Martinian, S. Yekhanin, and J. S. Yedidia, "Secure biometrics via syndromes," *Proc. Allerton Conf. on Commun., Control and Comp.*, 2005.

[5] R. Kötter, *On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes*, Ph.D. dissertation, Dept. of Elec. Engr., Linköping University, Linköping, Sweden, 1996.

[6] K. Lee and M. O'Sullivan, "An interpolation algorithm using Gröbner bases for soft-decision decoding of Reed-Solomon codes," *Proc. IEEE Intl. Symp. Info. Theory*, pp. 2032-2036, Jul. 2006.

[7] M. Sudan, Decoding of Reed-Solomon codes beyond the error correction bound," *J. Complexity*, 13:180C193, 1997.

[8] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Info. Theory*, vol. 49, no. 11, pp. 2809-2825, Nov. 2003.

[9] J. Zhu and X. Zhang, "Efficient VLSI architecture for soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Circits and Syst.-I*, vol. 55, no. 10, pp. 3050-3062, Nov. 2008.

[10] R. Kötter, J. Ma and A. Vardy, "The re-encoding transformation in algebraic list-decoding of Reed-Solomon codes," *IEEE Trans. Info. Theory*, vol. 57, no. 2, pp. 633-647, Feb. 2011.

[11] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[12] R. M. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Trans. Info. Theory*, vol. 46, no. 1, pp. 246-257, Jan. 2000.

[13] X. Zhang and K. K. Parhi, "Fast factorization architecture in soft-decision Reed-Solomon decoding," *IEEE Trans. VLSI Syst.*, vol. 13, no. 4, pp. 413-426, Apr. 2005.