The mathematics of Minisketch sketches

This is an unconventional mathematical overview of the PinSketch algorithm without references to coding theory[1].

Set sketches

A sketch, for the purpose of this description, can be seen as a "set checksum" with two peculiar properties:

- Sketches have a predetermined capacity, and when the number of elements in the set is not higher than the capacity, minisketch will always recover the entire set from the sketch. A sketch of *b*-bit elements with capacity *c* can be stored in *bc* bits.
- The sketches of two sets can be combined by adding them (XOR) to obtain a sketch of the symmetric difference between the two sets (*i.e.*, all elements that occur in one but not both input sets).

This overview explains how sets can be converted into a sketch and how a set can be recovered from a sketch.

From field elements to sketches

Data entries as field elements

Every integer in the range $[1 \ldots 2^b-1]$ (the acceptable data elements for a Minisketch sketch with field size *b*) can be mapped to a nonzero field element of $GF(2^b)$. In this finite field, we can add and multiply elements together, with many of the expected properties for those operations. Addition (and subtraction!) of field elements corresponds to bitwise XOR of the integers they correspond to, though multiplication is more involved.

Sets as power series

We define a function *S* which maps field elements *m* to the following formal power series (similar to a polynomial, except there can be an infinite number of terms, and we don't care about concepts like convergence as we're never going to actually evaluate it for a specific value of *x*):

- $S(m) = 1 + mx + m^2x^2 + m^3x^3 + \ldots$.

We then extend this function to operate on sets of field elements, by adding together the images of every set element. If $M = \{m_1, m_2, \ldots \}$:

- $S(M) = S(\{m_1,m_2,\ldots\}) = S(m_1) + S(m_2) + \ldots = (1 + 1 + \ldots) + (m_1 + m_2 + \ldots)x + (m_1^2 + m_2^2 + \ldots)x^2 + (m_1^3 + \ldots$

Because in our field addition corresponds to XOR of integers, it holds for every *a* that $a + a = 0$. This carries over to the *S* function, meaning that $S(a) + S(a) = 0$ for every *a*. This means that the coefficients of these power series have the second of the properties we desire from a sketch, namely that an efficient operation exists to combine two sketches such that the result is a sketch of the symmetric difference of the sets. It holds that $S(\{m_1,m_2\}) + S(\{m_2,m_3\}) = S(m_1) + (S(m_2) + S(m_2)) + S(m_3) = S(m_1) + S(m_3) = S(\{m_1,m_3\})$. The question is whether we can also efficiently recover the elements from their power series' coefficients.

An infinity of coefficients is hard

To make reasoning about these power series easier, notice that the series for a single element is in fact a geometric series. If we were working over real numbers rather than a finite field and $|mx| < 1$*, it would converge to *$(1 - mx)^{-1}$. Convergence has no meaning in formal power series, however it is still the case that:

- $(1 - mx)\, S(m) = 1$

You can verify this by seeing that every coefficient except the constant one gets cancelled out by the multiplication. This can be generalized to the series for multiple set elements. For two elements we have:

- $(1 - m_1x)\ (1 - m_2x)\ S(\{m_1,m_2\}) = (1 - m_1x)\ (1 - m_2x)\ (S(m_1) + S(m_2)) = (1 - m_2x) + (1 - m_1x)$

And for three:

- $(1 - m_1x)\ (1 - m_2x)\ (1 - m_3x)\ S(\{m_1,m_2,m_3\}) = (1 - m_1x)\ (1 - m_2x)\ (1 - m_3x)\ (S(m_1) + S(m_2) + S(m_3)) = (1 - m_2x)\ (1 - m_3x) + (1 - m_1x)(1 - m_3x) + (1 - m_1x)(1 - m_2x)$

In each case, we notice that multiplying $S(M)$ with $(1 - m_ix)$ for each element $m_i \in M$ results in a polynomial of degree $n-1$.

Solving for the set elements

The above insight lets us build a solver that extracts the set elements from the coefficients of a power series. If we can find a polynomial L that is the product of n different $(1 - m_ix)$ factors for various values of m_i, such that $P = S(M)L$ is an $n-1$ degree polynomial, then those values m_i are the elements of M.

The coefficients of P are nontrivial expressions of the set elements themselves. However, we can just focus on the coefficients of degree n and higher in P, as those are all 0. Let s_i be the coefficients of $S(M)$, and l_i the coefficients of L. In other words, $S(M) = s_0 + s_1x + s_2x^2 + s_3x^3 + \ldots$ and $L = l_0 + l_1x + l_2x^2 + l_3x^3 + \ldots + l_nx^n$. Note that $l_0 = 1$, as it is the product of all the 1 terms in the $(1 - m_ix)$ factors.

Here are the equations for the coefficients of $S(M)L$ of degree $n+1$ through $2n$:

- $s_{n+1} + s_{n+0}l_1 + s_{n-1}l_2 + s_{n-2}l_3 + \ldots + s_1l_n = 0$
- $s_{n+2} + s_{n+1}l_1 + s_{n+0}l_2 + s_{n-1}l_3 + \ldots + s_2l_n = 0$
- $s_{n+3} + s_{n+2}l_1 + s_{n+1}l_2 + s_{n+0}l_3 + \ldots + s_3l_n = 0$
- ...
- $s_{2n} + s_{2n-1}l_1 + s_{2n-2}l_2 + s_{2n-3}l_3 + \ldots + s_nl_n = 0$

These are n linear equations with n unknowns (the l_i values, for $i=1..n$), which can be solved using Gaussian elimination. After doing so, we have the coefficients of L, which can then be factored into first degree factors of the form $(1 - m_ix)$. The resulting m values are our set elements.

Putting it all together

Interestingly, only $2n$ coefficients of $S(M)$ were needed for solving the set of equations above. This means we have our answer: the coefficients 1 through $2n$ of $S(M)$, or the list $[m_1 + m_2 + \ldots, m_1^2 + m_2^2 + \ldots, \ldots, m_1^{2n} + m_2^{2n} + \ldots]$ functions as a sketch, satisfying the two properties we want:

- Sketches can be combined to form the sketch of their symmetric difference, by simply pairwise adding the list elements together.
- With $2n$ list elements we can efficiently recover n elements from a sketch.

Capacity and difference

The approach above only works when the number of elements n in the sketch is known. Of course we want to support cases where only an upper bound on the number of elements in the sketch is known, the capacity c. Given that we can reconstruct a set of size c from a sketch with $2c$ terms, we should be able to reconstruct a set of size n too as long as $n \leq c$. This is simply a matter of trying to solve the above set of equations assuming values of n that count down from c until a solution is found for one. This is known as the Peterson-Gorenstein-Zierler algorithm.

Optimizations

Halving the sketch size

We can in fact only include the odd terms in the sketch, and reconstruct the even ones before solving the equation to find L. This means the size of a sketch becomes just c field elements, the same size as would be needed to send its contents naively.

To see how this is possible, we need the Frobenius endomorphism, which in short states that in fields where $x + x = 0$ it holds that $(x + y)^2 = x^2 + y^2$ for every x and y (the dream of every high school math student!). This means that:

- $s_2 = m_1^2 + m_2^2 + ... = (m_1 + m_2 + ...)^2 = s_1^2$.
- $s_4 = m_1^4 + m_2^4 + ... = (m_1^2 + m_2^2 + ...)^2 = s_2^2$.
- $s_6 = m_1^6 + m_2^6 + ... = (m_1^3 + m_2^3 + ...)^2 = s_3^2$.
- ...

In other words, we only need to send $s_1, s_3, s_5, ..., s_{2n-1}$ to recover all $2n$ s_i values, and proceed with reconstruction.

Quadratic performance rather than cubic

Using Gaussian elimination to solve the set of equations above for the l_i values requires $O(n^3)$ field operations. However, due to the special structure in the equations (look at the repeated s_i values), it can be solved in $O(n^2)$ time using a number of techniques, including the Berlekamp-Massey algorithm (BM).

Roots instead of factorization

As explained above, the polynomial L can be factored into $(1 - m_i x)$ factors, where the values m_i are the set elements. However, since we know that a decodable sketch must result in a polynomial that is fully factorizable into degree-1 factors, we can instead use a more efficient root-finding algorithm rather than a factorization algorithm. As the root of each $(1 - m_i x)$ factor is m_i^{-1}, we conclude that the set elements are in fact the inverses of the roots of L.

Avoiding inversions

As inversions are a relatively expensive operation, it would be useful to avoid them.

Say that we're trying to find the inverses of the roots of $L = 1 + l_1 x + l_2 x^2 + ... + l_n x^n$, then we're really interested in the solutions y for $1 + l_1 y^{-1} + l_2 y^{-2} + ... + l_n y^{-n} = 0$. By multiplying both sides in the equations with y^n, we find $l_n + l_{n-1} y + l_{n-2} y^2 + ... + y^n = 0$.

In other words, we can find the inverses of the roots of L by instead factoring the polynomial with the coefficients of L in reverse order.

- [1] For those familiar with coding theory: PinSketch communicates a set difference by encoding the set members as errors in a binary BCH codeword 2^{bits} in size and sends the syndromes. The linearity of the syndromes provides all the properties needed for a sketch. Sketch decoding is simply finding the error locations. Decode is much faster than an ordinary BCH decoder for such a large codeword because the need to take a discrete log is avoided by storing the set in the roots directly instead of in an exponent (logically permuting the bits of the codeword).