DRAFT! - UNDER PEER REVIEW AND WORK

Bitcoin and ethereum both use elliptic-curve cryptography for generating keys and signing transactions.
When work started on eth 2.0 (transition to POS and sharding), all of a sudden simple ECDSA signatures just didn't cut it anymore as the number of signatures to verify was too big and would take too long on an average machine (See Justin Drake's post).
Signature aggregation was considered, specifically BLS signatures. This will enable aggregating multiple signatures into one (fixed size), verification also speeds up significantly as a result.
That's a mouth-full of terminology but what it enabled was an efficient way to support a very large amount of eth 2.0 validators working in many committees every epoch.

This blog post tries to break down BLS signatures and explain how it works behind the scene.
Wonderful sources of information i've used are Craig Costello's Pairing for Beginners and Steve Diaz's paper
A prerequisite for this article is a good knowledge of how elliptic-curve cryptography works.

BLS Signatures - overview
BLS stands for Boneh-Lynn-Shacham, it's a signature scheme that is based on bi-linear pairings. A pairing is a bilinear map of 2 groups G1, G2 in some other group, Gt.
The relationship between G1, G2 and Gt will be clear later on.

Pairings - naive example
Elliptic-curves have some basic operations we can do on curve points, namely addition and scalar multiplication.

- Point addition: P+Q=R
- Scalar multiplication: nP=P+P+..+P

Bi-linear pairings go a step forward and enable us to perform more complicated operations.
For $P \in G_1$ and $Q \in G_2$, bilinearity e means:
$e([a]P, [b]Q) = e(P, [b]Q)^a = e([a]P, Q)^b = e(P, Q)^(ab) = e([b]P, [a]Q)$

From the above we can infer that to verify a BLS signature, we have

- Secret key sk
- Public key (pk) sk*g1
- sig = sk*H(m)
- hash-to-point H(m)

Verifying the signature is simple a 2 pairing equation $e(g_1, sig) = e(pk, H(m))$ or if expanded: $e(g_1, H(m) * pk) = e(g1 * sk, H(m)) = e(g1, H(m))^(sk)$
Vitalik gave a great simple and intuitive of pairings in his article:
If $e(X, Y) = 2^(XY)$, then the pairing looks like:

e(3,4+5) = e(3,4) * 3(3,5) = 2^(3*4)*2^(3*5) = 2^27
Of Course the above is not really useful for real world use cases.
To better understand how pairings work, we will need to cover some basic concepts, to be fair those are fairly complicated but necessary to understand the magic behind BLS signatures.

Point at infinity
A point at infinity is an important concept in elliptic curve math. Its basic definition is a point which is both infinitely high and low in the y direction. We can also imagine the point at infinity as the identity of the group (a special type of element of a set with respect to a binary operation on that set, which leaves any element of the set unchanged when combined with it).

$$P \oplus \mathcal{O} = P.$$

Group points have an order, a number N that if multiplied by the point will "send" the point to infinity.
Example: $E/F101 : y^2 = x^3 + x + 1$, group order #E($F_q$)=105, generator(g)=(47,12).
Lagrange's theorem says that to get a point's order we simply multiply it by the cofactor $h = (e)/r$. A point of order 3 is [105/3](47,12)=(28,8). There also exists a point of order 1 which is [105](47,12) = point at infinity. The point's order also represents it's r-torsion.
R-tortions are extremely important in defining G1 and G2 as we will see below.

Divisors
At the heart of elliptic curve pairings lies the relationship between the curve and functions which intersect it.
A divisor is a way of representing a set of point on curve E

$$D = \sum_{P \in E(\bar{\mathbb{F}}_q)} n_P(P),$$

The () around the P indicates it's a sum in divisor.

Better yet, a divisor is a way to count zeros and infinities of a function on curve E.
Let's take an example curve $E/F23 : y^2 = x^3 + 5x + 7$ and the tangent l': $y = 4x + 20$.
We would like to find the divisor (l').
Substituting l' in E will result in $(4x + 20)^2 = x^3 + 5x + 7$, simplifying the equation will give us:
$x^3 + 7x^2 + 6x + 21 = 0$ (eq 1)

(remember we are dealing with modular arithmetics so the rules of multiplication, addition and subtraction are a bit different)
We can factorize eq 1 to $(x-2)^2(x-12)=0$.
We know that a line intersects our curve E in 3 points but since this is a tangent we will count that point twice. Our tangent intersects E at P(2,5) with multiplicity 2 and at Q(12,1) with multiplicity 1 (note that Q is equal to [2]P).
The divisor of l' = 2(P)+(Q)-3(O) = 2(P)+[2]P-3(O)

Divisors can be multiplied and divided like functions are. For multiplication we simply add up all the divisor points (with multiplicities) and for division we simply subtract.
Divisors have a concept of degree which is simply adding up the multiplicities.
If (f)=3(P) + 5(R) it's Deg((F)) = 8.
Divisors can be added together like so:

$$\sum_{P\in E} n_P(P) + \sum_{P\in E} m_P(P) = \sum_{P\in E} (n_P + m_P)(P).$$

They also have a concept of support, $supp(D) = P \in E : np <> 0$

With the above in mind, we can think of divisors as sub-groups. Meaning, divisors can be batched up into defined groups depending on their properties.

- Degree zero divisors $Div^0(E) is D \in Div(E), Deg(D) = 0$
- If a divisor D on E=(f), f is a function. It's called a principal divisor, Prin(E).
- $Prin(E) \in Div^0(E) \in Div(E)$

Divisor can be called equivalent if $D_1 - D_2 = (f) \in Prin(E)$, in other words, 2 divisors are equivalent if the difference between them is the zeros and poles of a rational function.
If D is a degree 0 divisor than for some point P it's equivalent to (P)-(O)

As we will see shortly, the importance of divisors in pairings is crucial as it deals with computing very large degree functions on E, and then evaluating these functions at other divisors.
Evaluating a function $f \in F_q(E)$ at a divisor $D = \sum_{P\in E} np(P)$ is defined as:

$$f(D) = \prod_{P\in E} f(P)^{n_P}.$$

Weil Pairing

Let $P, Q \in E(F_{q^k}[r])$ (in plain english 2 points on curve E over a field F_{q^k} for the r-torsion group) and D_P, D_Q are degree zero divisors with disjoint supports such that D_P ~ (P)-(O), D_Q ~ {Q)-(O), there exists functions f, g such that $(f) = rD_p$ and $(g) = rD_Q$.
Weil pairing is defined as:

$$w_r(P,Q) = \frac{f(D_Q)}{g(D_P)}.$$

Basically the above means that we take 2 points and map them to another group. This relationship and the way it is calculated gives us the bilinearity as described above:
$e([a]P,[b]Q) = e(P,[b]Q)^a = e([a]P,Q)^b = e(P,Q)^(ab) = e([b]P,[a]Q)$

Let's see a naive example using the same curve, $G_1 X G_1 = G_T$
(example 2.4 page 13 from Steve Diaz's paper).

Let S, T ∈ E[n]. Let D_S and D_T be divisors of degree 0 such that sum(D_S) = S and sum(D_T) = T and such that D_S and D_T have no points in common. Let f_S and f_T be functions such that div(f_S) = nD_S and div(f_T) = nD_T .
Find $e_n(S,T)$
A natural choice of divisors is D_S =[S]−[∞],D_T =[T+R]−[R].
$E/F_7 : y^2 = x^3 + 2$
P=(0,3),T=(5,1) such that $D_{(0,3)}$ =[(0,3)]−[∞],
$D_{(5,1)}$=[((5,1)+(6,1))]−[(6,1)]=[(3,6)]−[(6,1)].
$f : y - 3$ and $g : (4x - y + 1)/5x - y - 1)$. $D(f) = 3D_P$ and $D(g) = 3D_T$
* For finding f, g we can use Miller's algorithm

Our next step is to calculate $f(D_Q), g(D_P)$.
$f(D_T) = f(3,6)/f(6,1) = (6-3)/(1-3) = 2(mod 7)$
$g(D_P) = 4(mod 7)$, The way we calculate the point to infinity is by transferring to projective coordinates and treating the point to infinity as (0,1,0)
Our pairing result is $e_n(S,T) = 4/2 = 2(mod 7)$

Bilinearity

Now comes the cool part, the way we evaluate divisor points at functions treats the multiplicity of a point (in the divisor) as an exponent, this means that if we would have

done this example for [2]P, the result would have been $e_n(S,T)^2$. That's some cool bilinearity!

The example above is of-course not very secure, usually the curve needs to be over a very large field and the r-torsion group needs to be very big as well.
This leads to an issue as D_P, D_T need to have disjoint support which can't be found in F_q anymore. That is why most secure implementations of pairings do not use symmetric groups ($G_1 X G_1$) but rather group extensions (think complex numbers).

Summary
BLS signatures are based on a concept called pairing which presents bilinearity between it's components. Those are just fancy words for saying that we can apply, for example, some multiplication on one of the components of the pairing and the result will be that base pairing result to the a.
This fact enables us to evaluate, via pairings, digital signatures. The nice characteristics of BLS signatures are that they are aggregatable and deterministic which opens up a lot of possibilities.
Two of those possibilities are:

- eth 2.0: as it is built today, using committee which aggregate signatures. This enables finality within 2 epochs (~13min)
- staking pools: as the signatures are deterministic and aggregatable it makes them ideal to use within an MPC setting of staking pools.