

Update_2.0

zk-SNARKs

* (This update is regarding zk-SNARKS implementation)

■ Vulnerabilities

High: 3

Middle: 3

Low: 2

Note: 1

■ Details

Priority : Note

Issue :

<https://github.com/quras-official/quras-anonymous-library/blob/master/Common/include/libsnark/gadgetlib1/protoboard.tcc#L131>

Invalid function call .. size_t i = 0; i < constraint_system.num_variables; ++i

Correct one should be .. size_t i = 0; i < constraint_system.num_variables(); ++i

```

120
121     template<typename FieldT>
122     bool protoboard<FieldT>::is_satisfied() const
123     {
124         return constraint_system.is_satisfied(primary_input(), auxiliary_input());
125     }
126
127     template<typename FieldT>
128     void protoboard<FieldT>::dump_variables() const
129     {
130         #ifdef DEBUG
131             for (size_t i = 0; i < constraint_system.num_variables; ++i)
132             {
133                 printf("%-40s --> ", constraint_system.variable_annotations[i].c_str());
134                 values[i].as_bigint().print_hex();
135             }
136         #endif
137     }
138
139     template<typename FieldT>
140     size_t protoboard<FieldT>::num_constraints() const
141     {
142         return constraint_system.num_constraints();
143     }

```

Priority : Minor

Issue :

<https://github.com/quras-official/quras-anonymous-library/blob/master/Common/include/libff/algebra/fields/fp2.tcc#L146>

Add zero checks for square roots

```

144
145 template<mp_size_t n, const bigint<n>& modulus>
146 Fp2_model<n,modulus> Fp2_model<n,modulus>::sqrt() const
147 {
148     Fp2_model<n,modulus> one = Fp2_model<n,modulus>::one();
149
150     size_t v = Fp2_model<n,modulus>::s;
151     Fp2_model<n,modulus> z = Fp2_model<n,modulus>::nqr_to_t;
152     Fp2_model<n,modulus> w = (*this)*Fp2_model<n,modulus>::t_minus_1_over_2;
153     Fp2_model<n,modulus> x = (*this) * w;
154     Fp2_model<n,modulus> b = x * w; // b = (*this)^t
155
156     #if DEBUG
157         // check if square with euler's criterion
158         Fp2_model<n,modulus> check = b;
159         for (size_t i = 0; i < v-1; ++i)
160         {
161             check = check.squared();
162         }
163         if (check != one)
164         {
165             assert(0);
166         }
167     #endif
168
169     // compute square root with Tonelli-Shanks
170     // (does not terminate if not a square!)
171
172     while (b != one)
173     {
174         size_t m = 0;
175         Fp2_model<n,modulus> b2m = b;
176         while (b2m != one)
177         {
178             /* invariant: b2m = b^(2^m) after entering this loop */
179             b2m = b2m.squared();
180             m += 1;
181         }
182     }

```

Priority : High

Issue : Moving away from assertion to exception as defined in src/common/assert_except.hpp to remove potential DOS vectors from verifier.

For example potential Denial of service can occur on

https://github.com/quras-official/quras-anonymous-library/blob/master/Common/include/libsnark/reductions/r1cs_to_qap/r1cs_to_qap.tcc#L206

for invalid inputs with gadget it terminates with proof generation.

Another example :

https://github.com/quras-official/quras-anonymous-library/blob/master/Common/include/libff/algebra/curves/alt_bn128/alt_bn128_pairing.cpp#L348-L350

Here “assert” should be replaced by “assert_except”

Priority : High

Issue : Allow parameters of the curve to bound for arithmetic in the finite field square root (FP / FP2) QURAS has used Tonelli-Shanks Algorithm which is unbounded by nature. If someone uses a compressed curve point to decompress it in that case sqrt does not terminate.

Implementation here :

<https://github.com/quras-official/quras-anonymous-library/blob/master/Common/include/libff/algebra/fields/fp.tcc#L747>

This will lead to a denial of services to the node.

```

fp.tcc  +  fp2.tcc  protoboard.tcc  JoinSplit.cpp  JoinSplit.h  Quras_snarks.cpp
Quras_snarks
721
722
723 template<mp_size_t n, const bigint<n>& modulus>
724 Fp_model<n,modulus> Fp_model<n,modulus>::sqrt() const
725 {
726     Fp_model<n,modulus> one = Fp_model<n,modulus>::one();
727
728     size_t v = Fp_model<n,modulus>::s;
729     Fp_model<n,modulus> z = Fp_model<n,modulus>::nqr_to_t;
730     Fp_model<n,modulus> w = (*this)^Fp_model<n,modulus>::t_minus_1_over_2;
731     Fp_model<n,modulus> x = (*this) * w;
732     Fp_model<n,modulus> b = x * w; // b = (*this)^t
733
734     #ifdef DEBUG
735         // check if square with euler's criterion
736         Fp_model<n,modulus> check = b;
737         for (size_t i = 0; i < v-1; ++i)
738         {
739             check = check.squared();
740         }
741         if (check != one)
742         {
743             assert(0);
744         }
745     #endif
746
747     // compute square root with Tonelli--Shanks
748     // (does not terminate if not a square!)
749
750     while (b != one)
751     {
752         size_t m = 0;
753         Fp_model<n,modulus> b2m = b;
754         while (b2m != one)
755         {
756             /* invariant: b2m = b^(2^m) after entering this loop */
757             b2m = b2m.squared();
758             m += 1;
759         }
760
761         size_t j = v-m-1;
762         w = z;
763         while (j > 0)
764         {
765             w = w.squared();
766             --j;

```

Priority : Medium

Issue : Memory is being allocated using OPENSSL malloc and is directly used to set ppmutex value. It is very critical to validate that there was no issue while allocating the memory else this can lead to a node crash.

A sample solution will be using it something like this :

```

ptr = CRYPTO_malloc(size);
if (ptr == NULL) {
    XYZerr(XYZ_F_NM, ERR_R_MALLOC_FAILURE);
} else {
    /// continue
}

```

https://github.com/quras-official/quras-anonymous-library/blob/67843c2dc9d714f760276190d618f5558bca516c/Quras_snarks/QurasModules/utlis/utlis.cpp#L126

```

// Init
class CInit
{
public:
    CInit()
    {
        // Init OpenSSL library multithreading support
        ppmutexOpenSSL = (CCriticalSection**)OPENSSL_malloc(CRYPTO_num_locks() * sizeof(CCriticalSection));
        for (int i = 0; i < CRYPTO_num_locks(); i++)
            ppmutexOpenSSL[i] = new CCriticalSection();
        CRYPTO_set_locking_callback(locking_callback);
    }

```

Priority : Medium

Issue : In case the input data is empty the encode decode functions will behave unexpectedly. The base58 functions should be modified to handle different input types.

```
1 reference
public static byte[] Decode(string input)
{
    BigInteger bi = BigInteger.Zero;
    for (int i = input.Length - 1; i >= 0; i--)
    {
        int index = Alphabet.IndexOf(input[i]);
        if (index == -1)
            throw new FormatException();
        bi += index * BigInteger.Pow(58, input.Length - 1 - i);
    }
    byte[] bytes = bi.ToByteArray();
    Array.Reverse(bytes);
    bool stripSignByte = bytes.Length > 1 && bytes[0] == 0 && bytes[1] >= 0x80;
    int leadingZeros = 0;
    for (int i = 0; i < input.Length && input[i] == Alphabet[0]; i++)
    {
        leadingZeros++;
    }
    byte[] tmp = new byte[bytes.Length - (stripSignByte ? 1 : 0) + leadingZeros];
    Array.Copy(bytes, stripSignByte ? 1 : 0, tmp, leadingZeros, tmp.Length - leadingZeros);
    return tmp;
}
```

```
/// <returns>返回编码后的字符串</returns>
1 reference
public static string Encode(byte[] input)
{
    BigInteger value = new BigInteger(new byte[1].Concat(input).Reverse().ToArray());
    StringBuilder sb = new StringBuilder();
    while (value >= 58)
    {
        BigInteger mod = value % 58;
        sb.Insert(0, Alphabet[(int)mod]);
        value /= 58;
    }
    sb.Insert(0, Alphabet[(int)value]);
    foreach (byte b in input)
    {
        if (b == 0)
            sb.Insert(0, Alphabet[0]);
        else
            break;
    }
    return sb.ToString();
}
```

Priority : Medium

Issue : Bloom filters should check for valid m & k values before generating seed

```
1 reference
public BloomFilter(int m, int k, uint nTweak, byte[] elements = null)
{
    this.seeds = Enumerable.Range(0, k).Select(p => (uint)p * 0xFBA4C795 + nTweak).ToArray();
    this.bits = elements == null ? new BitArray(m) : new BitArray(elements);
    this.bits.Length = m;
    this.Tweak = nTweak;
}
```

Priority : High

Invalid ECC comparisons and generation, here a point is returned for an invalid curve also , the comparison is also faulty.

<https://github.com/quras-official/quras-blockchain-csharp/blob/master/QurasCore/Cryptography/ECC/ECPoint.cs#L30>

The validity of curve is not taken into consideration

```
7 references
internal ECPoint(ECFieldElement x, ECFieldElement y, ECCurve curve)
{
    if ((x != null && y == null) || (x == null && y != null))
        throw new ArgumentException("Exactly one of the field elements is null");
    this.X = x;
    this.Y = y;
    this.Curve = curve;
}

/// <summary>
```

```
/// <summary>
/// Compare with another object
/// </summary>
/// <param name="other">Another object</param>
/// <returns>Return the result of the comparison</returns>
34 references
public int CompareTo(ECPoint other)
{
    if (ReferenceEquals(this, other)) return 0;
    int result = X.CompareTo(other.X);
    if (result != 0) return result;
    return Y.CompareTo(other.Y);
}
```

Priority : Low

System will always return true for little endian

<https://github.com/quras-official/quras-anonymous-library/blob/master/Common/include/libff/common/utis.cpp#L93>

Here the pointer comparison will not point to equality of *c and leftmost 8 bits.

```
bool is_little_endian()
{
    uint64_t a = 0x12345678;
    unsigned char *c = (unsigned char*)&a;
    return (*c == 0x78);
}
```



Confidential