PALISADE

HOMOMORPHIC ENCRYPTION FOR PALISADE USERS

WEB & JAVASCRIPT APPLICATIONS

June 11, 2021

AGENDA

- Introducing the JavaScript/WebAssembly port
- Browser example (simple integers)
- Performance Review
- Javascript (Node.js)/WebAssembly examples
  - Threshold FHE
  - Proxy Re-Encryption
  - Buffer-based serialization
- Considerations and Future Work

JavaScript/WebAssembly Port

Performing homomorphic operations in Browser or NodeJS

Introducing the Javascript / WebAssembly port

- https://gitlab.com/palisade/palisade-wasm

- We are developing an npm package that provides access to the Palisade homomorphic encryption library to JavaScript / Typescript users
  - Currently in alpha, API is currently subject to change

- Currently optimized for the following schemes
  - BGVrns
  - CKKS
  - FHEW

- WebAssembly environment is typically limited to 4GB RAM

Using the JavaScript/WebAssembly Port

boolean.js

examples > js > binfhe > boolean.js > main

```javascript
// follows boolean.cpp example
async function main() {
  const factory = require('../../../lib/palisade_binfhe')
  const module = await factory()
  const cc = new module.BinFHEContext()
  cc.GenerateBinFHEContext(
    module.BINFHEPARAMSET.STD128, module.BINFHEMETHOD.GINX);
  console.log('Starting to generate keys.')
  const sk = cc.KeyGen()
  cc.BTKeyGen(sk)
  console.log('Completed generating keys.')

  console.log('Encrypting...')
  const ct1 = cc.Encrypt(sk, 1)
  const ct2 = cc.Encrypt(sk, 2)

  console.log('Computing...')
  const ctAND1 = cc.EvalBinGate(module.BINGATE.AND, ct1, ct2)
  const ct2Not = cc.EvalNOT(ct2)
  const ctAND2 = cc.EvalBinGate(module.BINGATE.AND, ct2Not, ct1)
  const ctResult = cc.EvalBinGate(module.BINGATE.OR, ctAND1, ctAND2)

  console.log('Decrypting...')
  const result = cc.Decrypt(sk, ctResult)
  console.log(
      'Result of encrypted computation of ' +
      '(1 AND 1) OR (1 AND (NOT 1) = ' + result)

  // since javascript cannot detect variables going out of scope reliably,
  // we must delete the crypto context manually
  cc.delete();
  return 0
}
main().then(exitCode => console.log(exitCode))
```

boolean.cpp 2

home > ted > git > palisade-development > src > binfhe > examples > boolean.cpp > ...

```cpp
#include "binfhecontext.h"

using namespace lbcrypto;
using namespace std;

int main() {
  auto cc = BinFHEContext();
  cc.GenerateBinFHEContext(STD128);
  auto sk = cc.KeyGen();

  std::cout << "Generating the bootstrapping keys..." << std::endl;
  cc.BTKeyGen(sk);

  std::cout << "Completed the key generation." << std::endl;
  auto ct1 = cc.Encrypt(sk, 1);
  auto ct2 = cc.Encrypt(sk, 1);
  auto ctAND1 = cc.EvalBinGate(AND, ct1, ct2);
  auto ct2Not = cc.EvalNOT(ct2);
  auto ctAND2 = cc.EvalBinGate(AND, ct2Not, ct1);
  auto ctResult = cc.EvalBinGate(OR, ctAND1, ctAND2);
  LWEPlaintext result;

  cc.Decrypt(sk, ctResult, &result);

  std::cout
      << "Result of encrypted computation of (1 AND 1) OR (1 AND (NOT 1)) = "
      << result << std::endl;

  return 0;
}
```


PALISADE

Supported Examples

- BINFHE
  - Boolean circuits
  - Boolean circuits with buffer-based serialization
- PKE
  - PRE buffer
  - Simple Integer
  - Simple Integer with BGVrns scheme
  - Simple Integer with buffer-based serialization
  - Simple Real Numbers
  - Threshold FHE

All examples demonstrated in previous Palisade webinars are supported by the WebAssembly Port

Naming / Documentation

- While the JS API docs are still a work in progress, the names in the JS port closely follow the underlying C++ functions (with some slight alterations)

C++	JS
CryptoContext<DCRTPoly>	CryptoContextDCRTPoly
EvalMult(Ciphertext,Ciphertext)	EvalMultCipherCipher
AND	BINGATE.AND

PALISADE

Capabilities

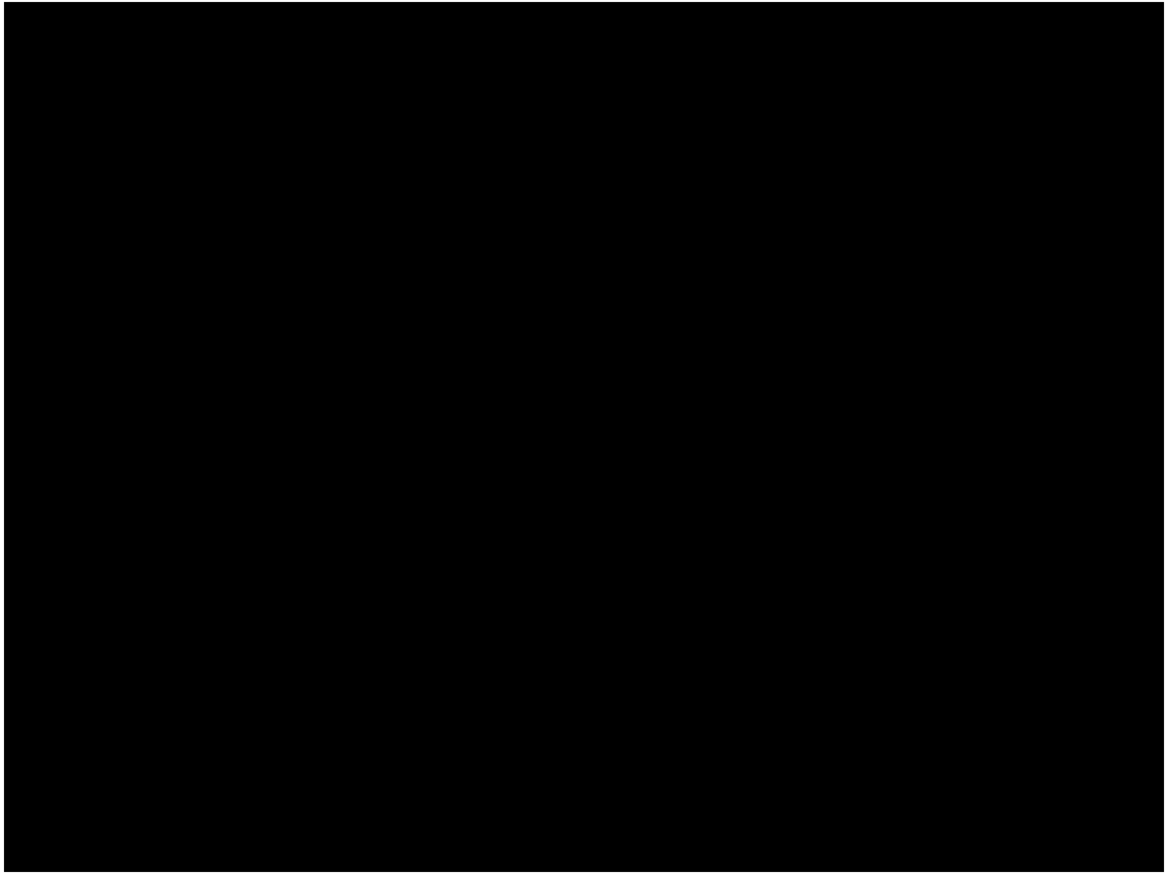
- All examples described in previous Palisade webinars can be re-created in JavaScript by using the WebAssembly port
- WebAssembly users can perform homomorphic encryptions over
  - Boolean circuits
  - Integers
  - Real Numbers
- Encryption keys can be serialized and deserialized to communicate over the network with peers
  - Since file access from WebAssembly is somewhat complex, only buffer serialization is currently supported

```
// Serialize cryptocontext in C++
Serial::SerializeToFile(
       DATAFOLDER + "/cryptocontext.txt",
       cryptoContext,SerType::BINARY);

// Serialize cryptocontext in JavaScript
const cryptoContextBuffer =
    module.SerializeCryptoContextToBuffer(
           cryptoContext,module.SerType.BINARY);
```


PALISADE

Examples: Web Demo

Simple Integers



Typescript bindings

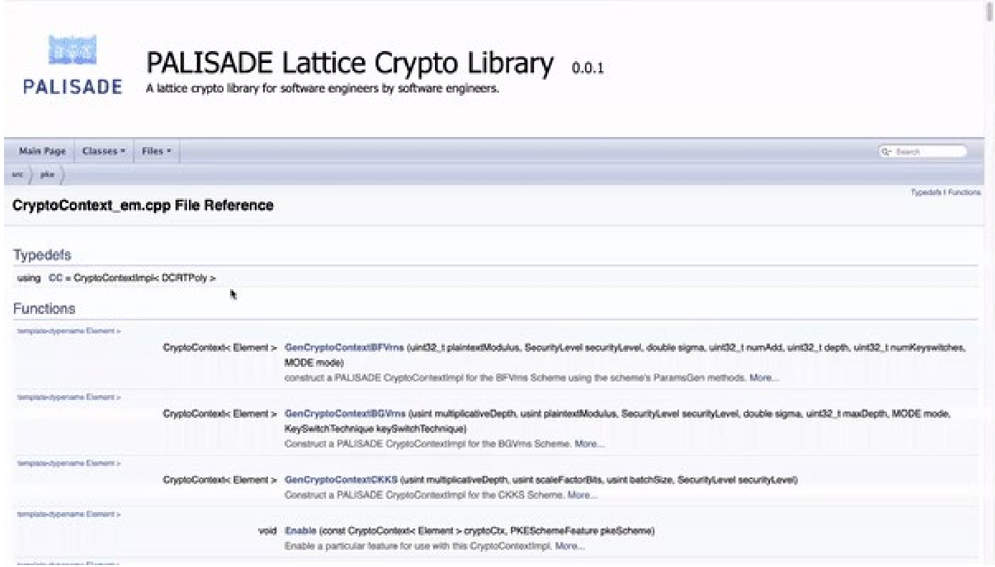
- The JavaScript port will also feature typescript bindings, which can catch errors at compile time rather than runtime
  - Also provides accurate autocomplete!

- This provides many of the structural guarantees of C++ while maintaining the flexibility of JavaScript’s OO and functional patterns

```
    const ctAND2 = cc.EvalBinGate(module.BINGATE.AND, ct2Not, ct1)
    const ctResult = cc.EvalBinGate(module.BINGATE.OR, ctAND1, ctAND2)

    cc.

    console.log('Decrypting...')
    const result = cc.Decrypt(sk, ctResult)
    console.log(
        'Result of encrypted computation of ' +
        '(1 AND 1) OR (1 AND (NOT 1)) = ' + result)

    // since javascript cannot detect variables going out of scope reliably,
    // we must delete the crypto context manually
    cc.delete();
    return 0
}
main().then(exitCode => console.log(exitCode))
```

Documentation

- **Doxygen for C/C++ to WASM:**



- **TypeDoc/TSDoc for WASM => Typescript (Coming soon):**

Performance Review

Performance

- **GCC**

```
bin/benchmark/lib-benchmark

Run on (8 X 4700 MHz CPU s)
CPU Caches:
  L1 Data: 32 KiB (x8)
  L1 Instruction: 32 KiB (x8)
  L2 Unified: 256 KiB (x8)
  L3 Unified: 12288 KiB (x1)
Load Average: 0.09, 0.39, 0.96
```

- **NodeJS WASM**

```
bin/lib-benchmark.js
```

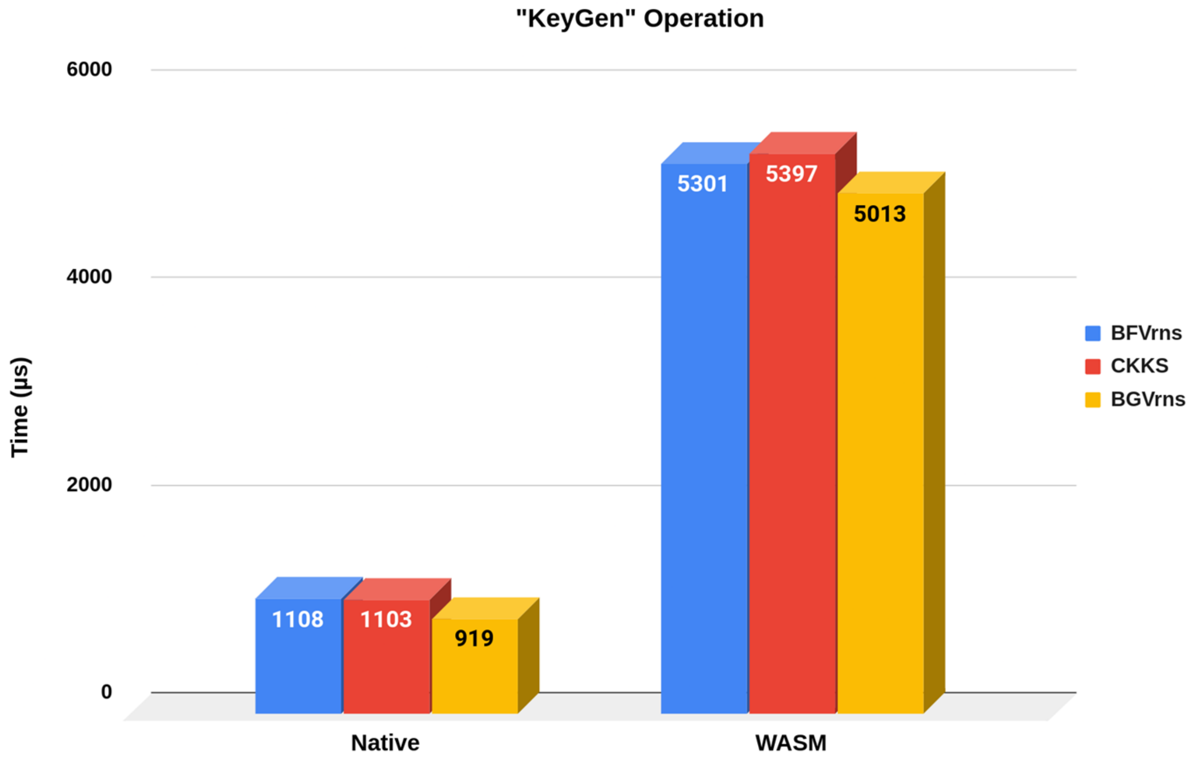
- **Clang**

```
└─ ./bin/benchmark/lib-benchmark
Unable to determine clock rate from sysctl: hw.cpufrequency: No such
2021-06-11 18:25:29
Running ./bin/benchmark/lib-benchmark
Run on (8 X 24.121 MHz CPU s)
CPU Caches:
  L1 Data 64 KiB (x8)
  L1 Instruction 128 KiB (x8)
  L2 Unified 4096 KiB (x8)
Load Average: 0.87, 1.18, 1.35
---------------------------------------------------------------------
Benchmark                          Time             CPU   Iterations
---------------------------------------------------------------------
NTTTransform1024                7.52 us         7.51 us        72559
INTTTransform1024               7.76 us         7.75 us        90139
NTTTransform4096                34.8 us         34.7 us        20047
INTTTransform4096               36.1 us         36.1 us        19437
NTTTransformInPlace1024         7.31 us         7.31 us        94614
INTTTransformInPlace1024        7.83 us         7.81 us        88619
NTTTransformInPlace4096         34.6 us         34.6 us        20398
INTTTransformInPlace4096        36.0 us         36.0 us        18888
BFVrns_KeyGen                   1789 us         1789 us          392
BFVrns_MultKeyGen               2797 us         2797 us          248
BFVrns_EvalAtIndexKeyGen        2947 us         2937 us          242
BFVrns_Encryption               1979 us         1979 us          349
BFVrns_Decryption                395 us          395 us         1781
BFVrns_Add                      14.5 us         14.4 us        48437
BFVrns_AddInPlace               7.98 us         7.98 us        84886
BFVrns_MultNoRelin              6222 us         6221 us          110
BFVrns_MultRelin                7009 us         7008 us          101
BFVrns_EvalAtIndex               648 us          648 us         1066
CKKS_KeyGen                     1972 us         1971 us          359
CKKS_MultKeyGen                 4558 us         4558 us          152
CKKS_EvalAtIndexKeyGen          4610 us         4608 us          152
```


PALISADE

Performance

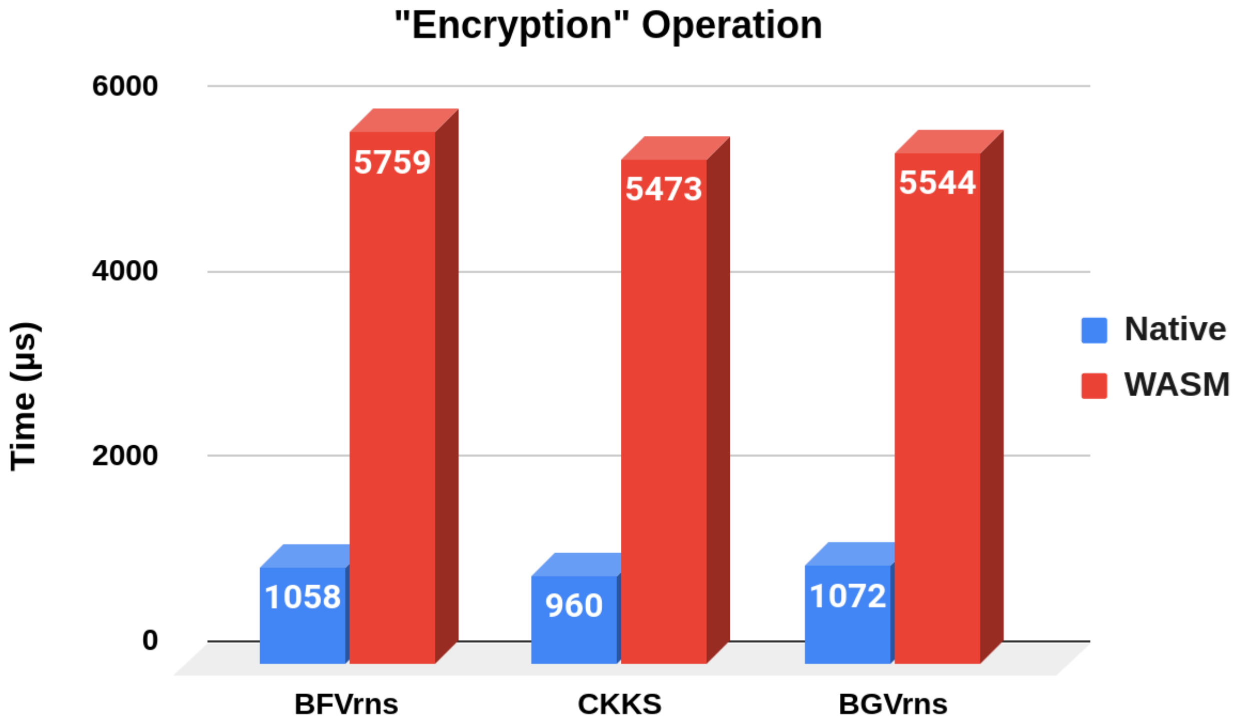
WebAssembly Performance Comparison per Scheme



PALISADE

Performance

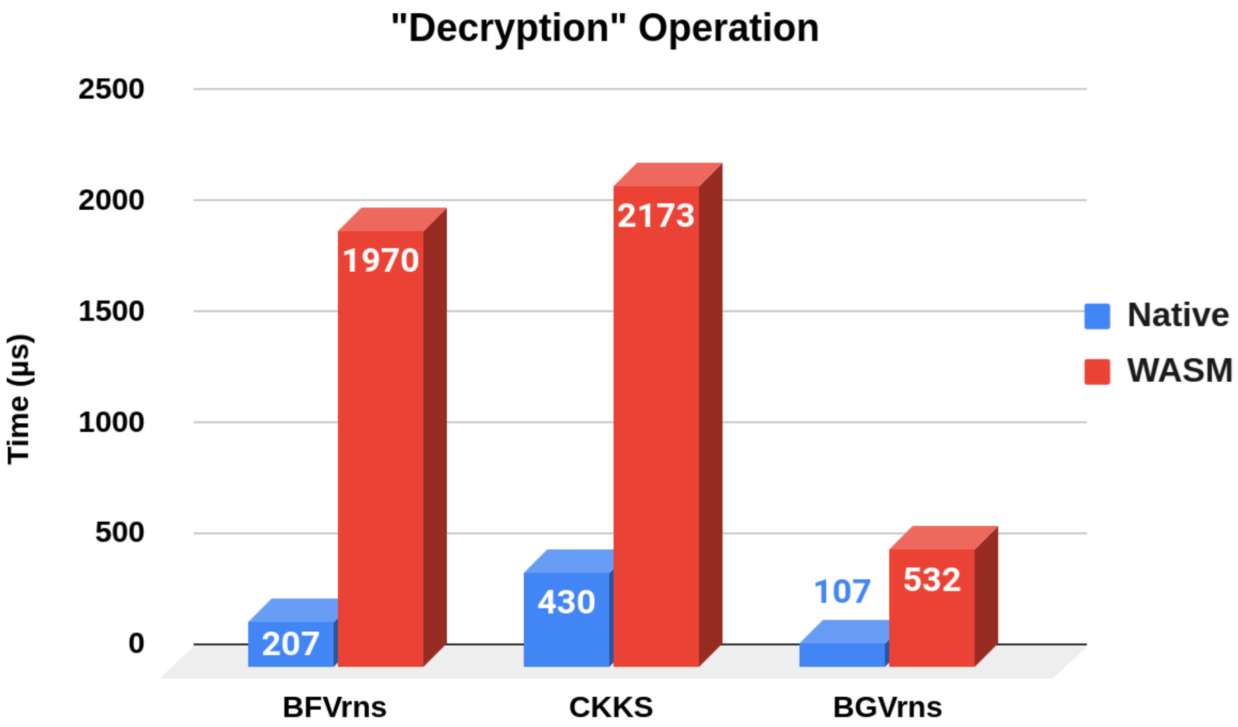
WebAssembly Performance Comparison per Scheme



PALISADE

Performance

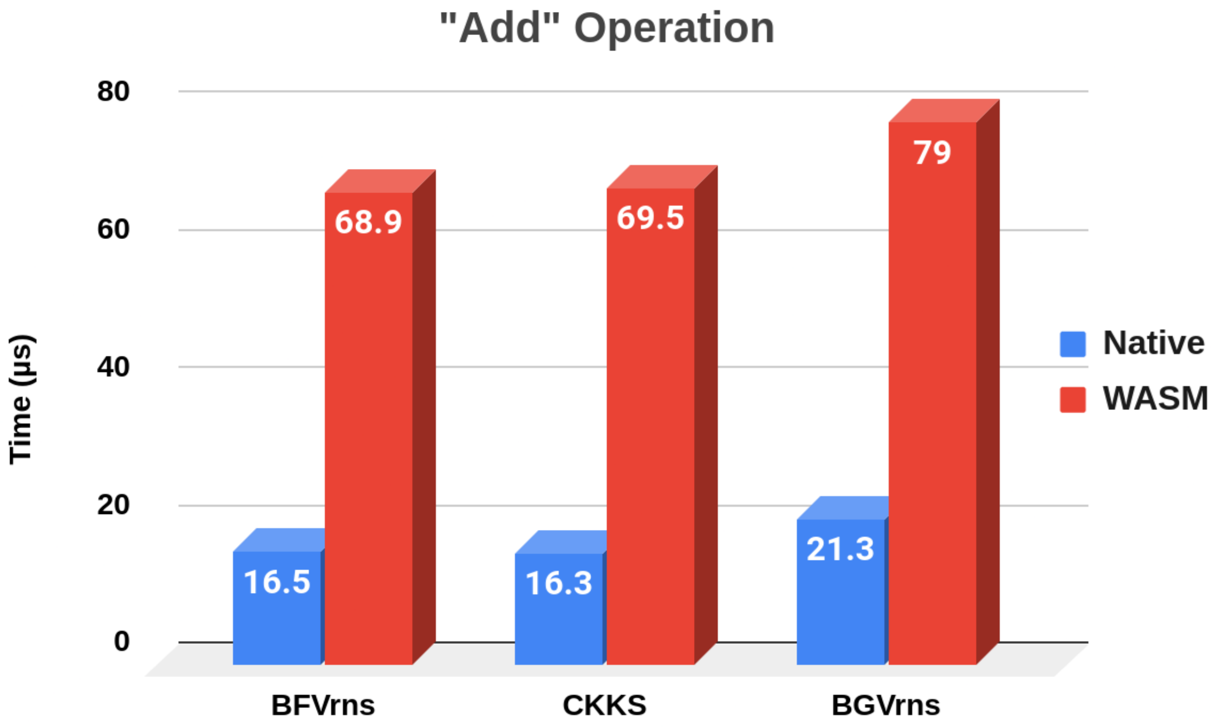
WebAssembly Performance Comparison per Scheme



PALISADE

Performance

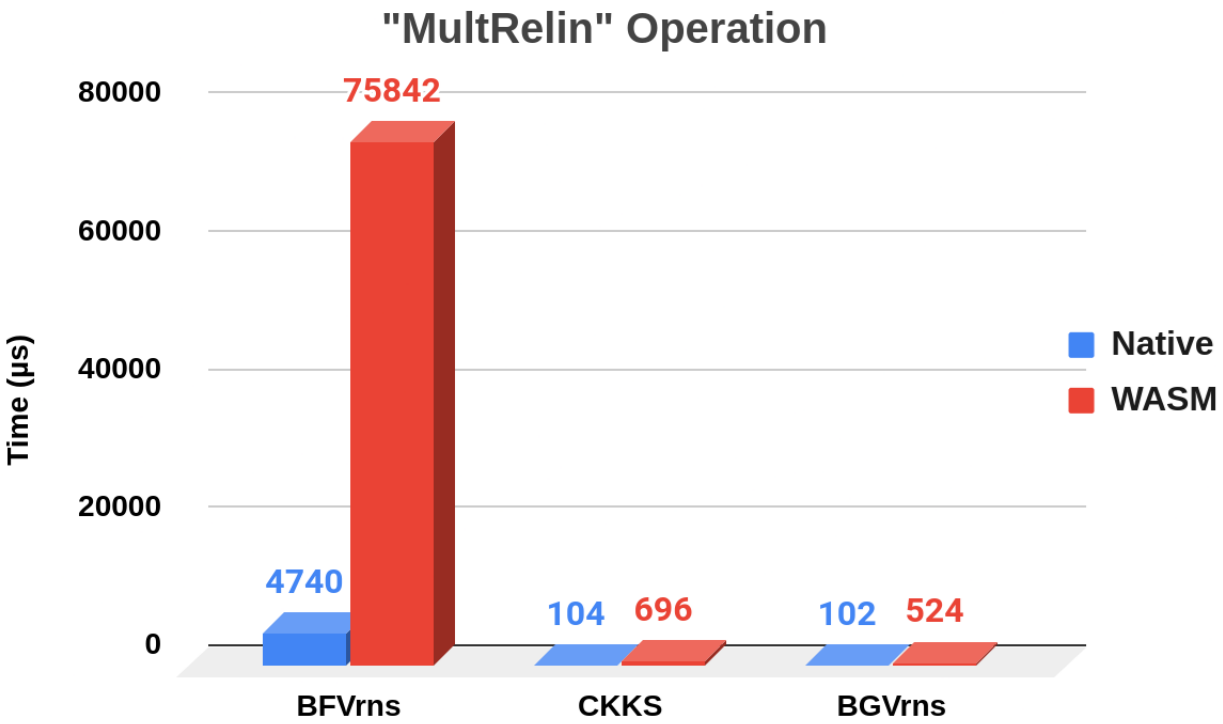
WebAssembly Performance Comparison per Scheme



PALISADE

Performance

WebAssembly Performance Comparison per Scheme

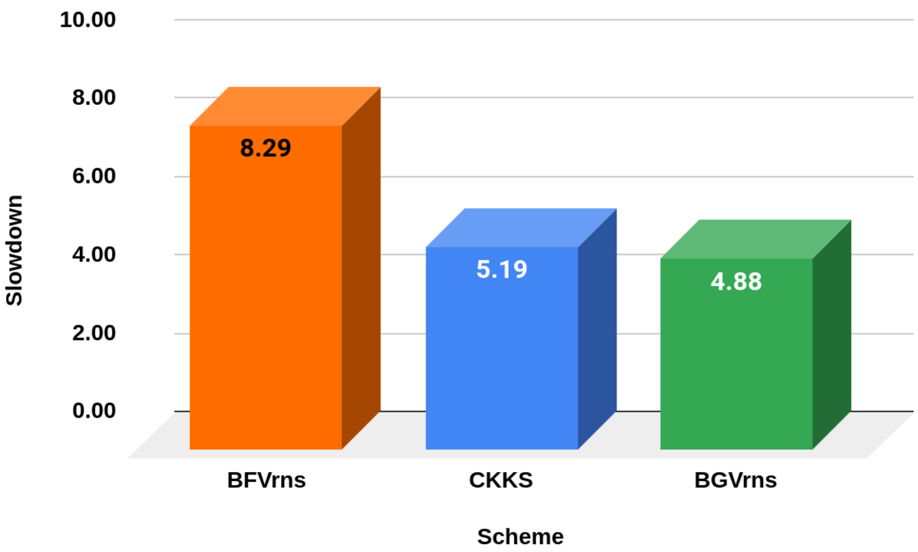


PALISADE

Performance

- For **integer calculations**, **BGVrns** is the preferred scheme
- For **real number calculations**, use **CKKS**
- BGVrns and CKKS are optimized for WebAssembly
- Multiplying/rotation of ciphertexts suffers approximately a **5X** slowdown with BGVrns, but receives a **20X** slowdown with BFVrns

WebAssembly Performance Comparison per Scheme



PALISADE

NodeJS Examples

Examples: Lives Demos

- Threshold FHE
- Proxy Re-Encryption
- Buffer-based serialization

PALISADE

Considerations and Future Work

Considerations of JavaScript/WebAssembly Port

- **Benefits**
  - Develop web-based homomorphic encryption applications with comparable performance when compared to native execution.

- **Limitations**
  - Currently generation of crypto context supports a limited number of parameters. This is work in progress.
  - Whereas C++ detects variables going out of scope, JavaScript users must explicitly call .delete() on every C++ handle they receive i.e. generated crypto context
  - OpenMP is not available for WebAssembly yet.
  - Very limited SIMD support mostly through emulation.
    Only the 128-bit wide instructions from AVX instruction set are available. 256-bit wide AVX instructions are not provided.

- **Alternatives**
  - Other binding options are available where users can write their own WASM bindings
  - **C++ addons** can also be used.

Future Work

- Optimize other schemes for WebAssembly

- Potential support for multi-threading by WebAssembly

- Examples to highlight potential use cases of the palisade-wasm
  - Develop web-based homomorphic solution
  - Networked Threshold-FHE, Proxy-ReEncryption

- We'd love more contributors. Please try Palisade-wasm and help us improve it.

- Please Help us spread the word!

THANK YOU!