Circomkit FFI

more than SnarkJS for your circuits

SoulForge zkBankai
https://soulforge.zkbankai.com/

Erhan Tezcan
https://github.com/erhant/circomkit-ffi

March, 2025

Circom

Circom is a hardware description language (HDL) specifically designed for creating **zero-knowledge proofs**.

It enables developers to create arithmetic circuits, which are then used to generate proofs that demonstrate a certain statement is true, without revealing any additional information about the inputs used to make that statement.

==>circom

CIRCUIT COMPILER

Circom

There are several steps considering an end-to-end zero-knowledge proof:

1. **Circuit** (written in Circom language)
2. **R1CS** (output of Circom compiler)
3. **Witness Calculator** (output of Circom compiler, in C or JS + WASM)
4. **Prover Key** & **Verification Key** (depending on protocol)

Using witness calculator, the (honest) prover computes a witness, and can verify the constraints via R1CS. Prover can then create a proof using the prover key and the witness.

A verifier with proof and verification key can verify the proof.

SnarkJS

Most Circom projects use SnarkJS for proof generation & verification, as it is the de-facto method as also given within Circom documentation.

However, SnarkJS is based on JavaScript & WASM and it comes with problems, especially for large circuits.
(https://hackmd.io/V-7Aal05Tiy-ozmzTGBYPA?view#Best-Practices-for-Large-Circuits)

There exist other prover backends, some based on Rust language (or has bindings to it) that could otherwise be used, but such alternatives require their own development effort.

Circomkit

Circomkit is a widely used (~70k downloads / year) toolkit that makes it much easier to develop & test Circom circuits, within a TypeScript environment.

It is both a library, and a CLI tool that can be used for all kinds of operations, from compiling circuits and generating proofs to verifying them or generating Solidity smart-contracts for verification.

It is open-source, available on:

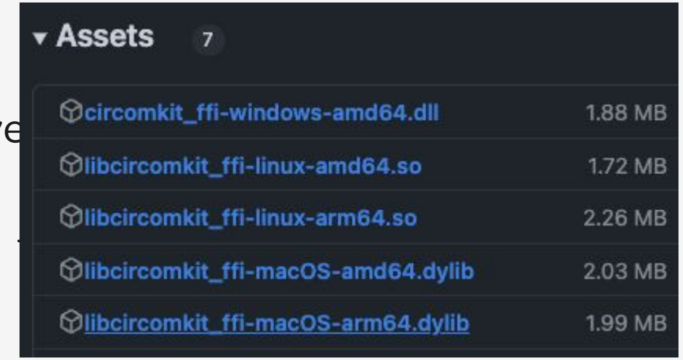
- https://github.com/erhant/circomkit
- https://www.npmjs.com/package/circomkit

Circomkit FFI

Circomkit FFI is a project that aims to close the development effort gap for alternative provers, dumbing down the entire process to just a single argument: *the prover name*!

We've developed a **Rust library** that is cross-compiled & released for **MacOS**, **Linux**, and **Windows** as static libraries. (`.dylib`, `.so`, `.dll` respectively).

Along with it, we have
TypeScript SDK published
that allows the user
this static library based on their platform.


▾ Assets 7

circomkit_ffi-windows-amd64.dll	1.88 MB
libcircomkit_ffi-linux-amd64.so	1.72 MB
libcircomkit_ffi-linux-arm64.so	2.26 MB
libcircomkit_ffi-macOS-amd64.dylib	2.03 MB
libcircomkit_ffi-macOS-arm64.dylib	1.99 MB

Circomkit FFI

As a quick preliminary, what is FFI? It stands for "*Foreign Function Interface*", which allows a **host** program to call a function on a **guest** program. In our case is, this is a **TypeScript program** calling a **dynamically linked Rust library**.

We do this by exposing C-like functions from our Rust library, which are non-mangled functions with C-string pointers as arguments, and a C-string pointer as return type.

The FFI functionality is given by libraries that we use, for Node and Bun respectively.

Circomkit FFI

We have integrated the following libraries:

- **Arkworks**: Groth16 for BN254
- **Lambdaworks**: Groth16 for BLS12-381
- **Ingonyama ICICLE**: Groth16 for BN254, *(experimental)*

The SDK supports two runtimes separately:

- **Bun** runtime, using `bun:ffi`
- **Node** runtime, using `ffi-rs`

Both the Rust library and TypeScript SDK are available on:

- https://github.com/erhant/circomkit-ffi
- https://www.npmjs.com/package/circomkit-ffi

Circomkit FFI

With the Circomkit FFI SDK, the entire process is as follows:

1. Create the SDK class, providing the library path.
2. Call the respective function with its args!

Node runtime, using `ffi-rs`

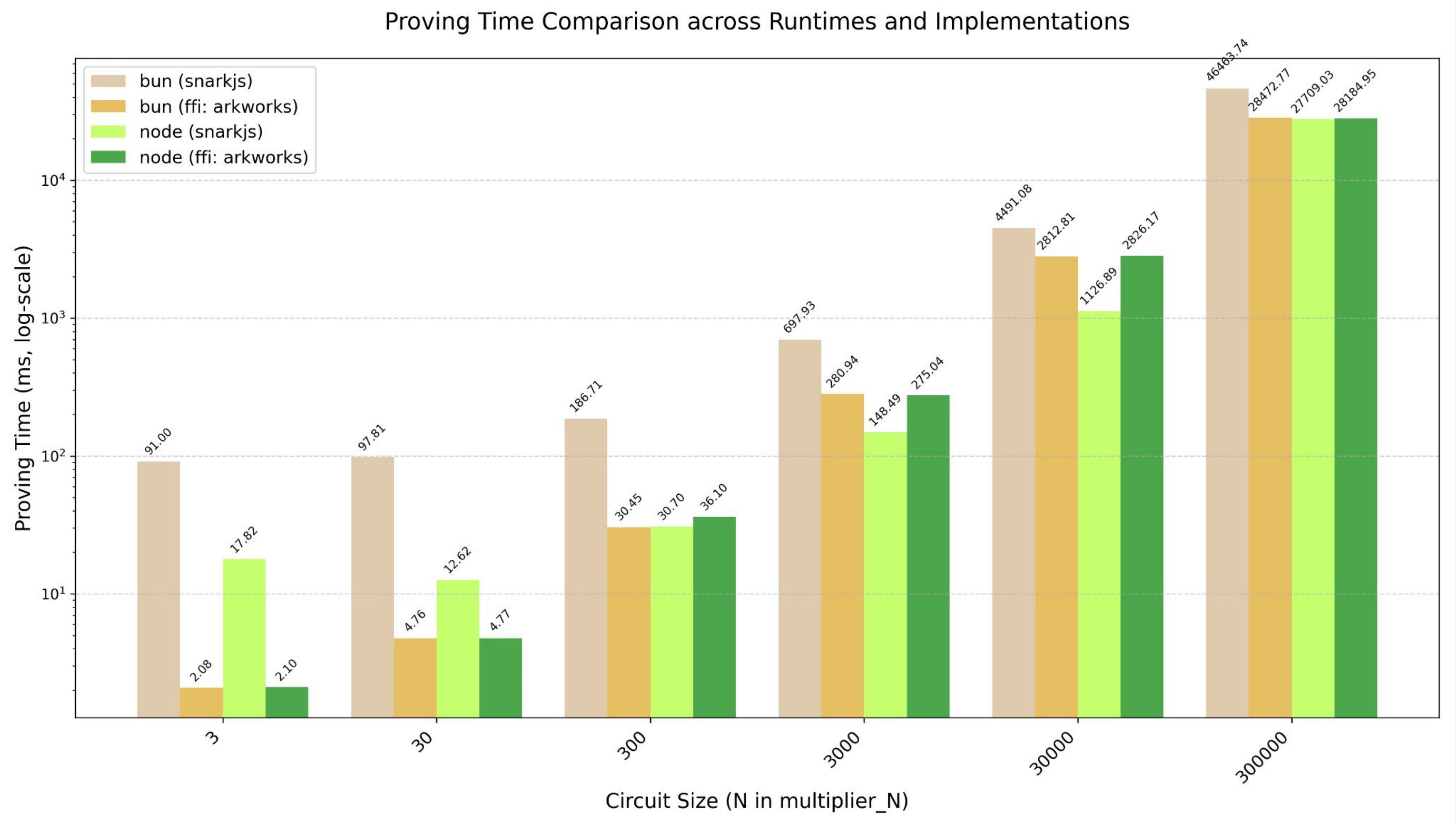
```
const circomkitFFI =
new CircomkitFFINode(libPath, open,
close, load);

const { proof, publicSignals } =
circomkitFFI.arkworks_prove(
  "witness.wnts",
  "circuit.r1cs",
  "prover_key.zkey"
);
```

Bun runtime, using `bun:ffi`

```
const circomkitFFI =
new CircomkitFFIBun(libPath);


const { proof, publicSignals } =
circomkitFFI.arkworks_prove(
  "witness.wnts",
  "circuit.r1cs",
  "prover_key.zkey"
);
```


Proving Time Comparison across Runtimes and Implementations
bun (snarkjs)
bun (ffi: arkworks)
node (snarkjs)
node (ffi: arkworks)
Proving Time (ms, log-scale)
10^1
10^2
10^3
10^4
91.00
2.08
17.82
2.10
97.81
4.76
12.62
4.77
186.71
30.45
30.70
36.10
697.93
280.94
148.49
275.04
4491.08
2812.81
1126.89
2826.17
46463.74
28472.77
27709.03
28184.95
3
30
300
3000
30000
300000
Circuit Size (N in multiplier_N)

Fin

https://github.com/erhant/circomkit-ffi

We thank Soulforge zkBankai for their support on this project!