The New Era of Blockchain Privacy

Paul Brody
EY Global Blockchain Leader
There isn’t much in the way of blockchain privacy today, though many are not bothered.

- Nearly all transaction data is readable in the clear
- Swaps, exchanges and mixers and liquidity pools all provide an element of pseudo-anonymity for individual users
- Large scale traders find their moves impossible to conceal
The lack of privacy hasn’t stopped the growth of some very powerful use cases.

The DeFi ecosystem is nearing $300bn in value
For many financial & industrial use cases, however, privacy is essential.

- Money and products are easily represented by digital tokens.
- Most enterprise assets are unique to the enterprise—so they're too easy to follow around on the blockchain.
- Enterprise purchases are not swaps—they are often complex smart contracts that contain unique and sensitive business information.
More than one kind of privacy tool is needed to unlock the universe of business applications

1 Asset Transfers and Payments
- Critical to keep what you’re buying and how much you are paying overall, as well as when you buy and where it goes a secret from your competition

2 Smart Contract Terms
- The contract logic – the major terms & conditions, are also sensitive information because they usually contain price and rebate information based on expected volumes
Once we can handle both assets and logic, we can handle pretty much every transaction.

1. **Tokens**
   - Asset tokens
   - Commodities
   - Currency tokens
   - Rights tokens
   - Insurance tokens
   - Loans and leases

2. **Contracts**
   - Procurement contracts
   - Loans
   - Royalties
   - Sales
   - Securities
   - Retail promotions

3. **Analytics**
   - Transaction history
   - Fraud detection
   - Product traceability
   - Tax liability
   - How do we gain insight from this process?

**Questions:**

- What items of value are being exchanged?
- What are the rules that govern this process?
Polygon Nightfall gets us down the first half of this path, around asset transfer privacy

- Once you have assets inside of Polygon Nightfall, there is no external visibility to how it moves to external viewers
- Assets can be put in and cashed out
- Auditable histories can be prepared but requires the cooperation of the sender or receiver
- Many new business models are unlocked by this

Polygon Nightfall is in production beta. Polygon Nightfall is a product of Polygon Technology, not EY. EY developed the original nightfall code and has contributed that code into the public domain. EY does not control or manage Polygon Nightfall and retains no ownership over the Nightfall code. Nightfall is a public domain, open-source initiative to which any person or firm can contribute and EY continues to contribute new ideas and code as well based on our own thinking about how privacy technology needs to develop to support widespread adoption. You can find the original Nightfall information at [https://github.com/eyblockchain/](https://github.com/eyblockchain/). If you want to learn about Polygon Nightfall, please visit [https://polygon.technology](https://polygon.technology)
EY OpsChain Supply Chain Manager is our first product that leverages privacy for industrial users.

<table>
<thead>
<tr>
<th>Raw materials</th>
<th>Manufacture</th>
<th>Transport</th>
<th>Warehouse</th>
<th>Sell</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Purchase raw materials</td>
<td>● Integrate items together into manufacturing output</td>
<td>● Put finished goods into an in-transit status</td>
<td>● Move into warehouse with a distributor</td>
<td>● Transfer to a retailer</td>
<td>● Build true end-to-end traceability for product history</td>
</tr>
<tr>
<td>● Create digital tokens to represent those assets</td>
<td>● New digital token incorporates the materials</td>
<td></td>
<td>● Unload container and truck</td>
<td>● Transfer ownership first to retailer and then to end customer</td>
<td></td>
</tr>
</tbody>
</table>

With privacy enabled, you can now track end-to-end multi-company value chains without disclosing sensitive business information to your competition.
Nightfall remains public domain and open source and we believe there is more work to be done.

It’s been a very long haul:

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
<th>Avg Per Tx Costs**</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>Work Started</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>Nightfall Prototype</td>
<td>$100</td>
</tr>
<tr>
<td>2019</td>
<td>Nightfall Version 2</td>
<td>$5</td>
</tr>
<tr>
<td>2020</td>
<td>Timber &amp; Batching</td>
<td>$2.5</td>
</tr>
<tr>
<td>2021</td>
<td>Nightfall Version 3</td>
<td>$1.00</td>
</tr>
<tr>
<td>2022</td>
<td>Polygon Nightfall Production Beta</td>
<td></td>
</tr>
</tbody>
</table>

**Transaction prices are approximate and will vary with network congestion and gas fees.

And there’s a lot more to do:

- Regulatory compliance tools
- Improved audit integrations
- Privacy-enabled swaps
- NFT theft protection
- Metadata masking

Polygon Nightfall is in production beta. Polygon Nightfall is a product of Polygon Technology, not EY. EY developed the original nightfall code and has contributed that code into the public domain. EY does not control or manage Polygon Nightfall and retains no ownership over the Nightfall code. EY does continue to contribute to the Nightfall code base and we develop new features in consultation with any member of the blockchain community that wishes to contribute. Other people and companies are free to contribute to this open source, public domain initiative. You can find the original Nightfall information at [https://github.com/eyblockchain/](https://github.com/eyblockchain/). If you want to learn about Polygon Nightfall, please visit [https://polygon.technology](https://polygon.technology).
The next big challenge is making business logic private, a task we’re focused on with Starlight.

- Starlight enables on-chain logic that’s not externally de-codeable.
- Business relationships with specific terms & conditions can then be applied without disclosing them to the wider public.
- Still subject to limitations of metadata “leakage” as transactions take place.
eCommerce didn’t take off without encryption and blockchain won’t scale without privacy.

- SSL certificates became available starting in 1994 on the Netscape browser.
- Prior to that time, online credit card transactions were done in the clear, leading to low levels of consumer trust.
- The actual rate of online fraud in the early days was very low
Thank you!

Paul R. Brody
Principal, Ernst & Young LLP
EY Global Blockchain Leader
paul.brody@ey.com

@pbrody
Starlight

Devcon VI

Chaitanya Konda
Applied Cryptographer, Blockchain
Agenda

1. Intro to Starlight
2. How does the transpiler work?
3. zApp Architecture
4. Example
Zero Knowledge Applications, or zApps are Awesome but …..

They have a **steep** learning curve

Experience with zero-knowledge proofs (ZKPs)

Take **time** to build

**Specialist** ZKP devs need to be hired

Starlight was born to solve all these problems.
Zero Knowledge Applications, or zApps are awesome but …..

Ordinary smart contract
myContract.sol

Decorated smart contract
myContract.zol

Transpiler

Complete, working zApp

Mark what you want private

Run ./zappify
How does transpiler work?

- **Solidity Smart Contract**
  
  ```solidity
  uint x;
  ```

- **Zolidity Smart Contract**
  
  ```solidity
  secret uint x;
  ```

- **zApp**
  
  - Contracts
  - Circuits

- **Backend Containers**
  
  - Timber
  - Tests
  - Blockchain Client
  - Zokrates

- zappify
Decorators – **Secret**

- **What it does?**
  - Contents of the variable remain confidential

- **For**
  - State variables
  - Function parameters
  - Functions (future enhancement)

- **Not for**
  - Local stack memory declarations

- **How it works?**
  - Create a commitment for this state variable that binds and hides the value

```solidity
contract Example {
  secret uint x; // owned by the contract deployer

  function add(secret uint y) public {
    known x += y;
  }
}
```
Decorators – **Known**

- **What it does?**
  - Only the secret state variable owner can update it
  - For incrementation statements of secret state variables

- **How it works?**
  - Proof of knowledge of existence of old commitment
  - Proof of knowledge of secret key of the public key in commitment
  - Nullifies old commitment
  - Create new commitment

```solidity
contract Example {

  secret uint x; // owned by the contract deployer

  function add(secret uint y) public {
    known x += y;
  }
}
```
Decorators – Unknown

- What it does?
  - Anyone can increment this secret state variable
- For
  - Incrementation statements of secret state variables
- How it works?
  - Create a new "part" commitment to hold only the value by which to update the amount
  - Secrete state variable is a partitioned variable whose value is a summation of all it’s “part” commitments

```solidity
contract Example {

  secret mapping(address => uint) balances;

  function deposit(uint amount) {
    balances[msg.sender] += amount;
  }

  function transfer(secret uint amount, secret address recipient) {
    balances[msg.sender] -= amount;
    unknown balances[recipient] += amount;
  }
}
```
Commitment structure

Normal State Variable
- Unique state ID applied by solc
- Value state represents ZKP public key of the owner
- Random salt

Mapping State Variable
- Secret mapping(address => uint256) balances;
- Mapping key (e.g., Alice’s address) or array index
- hash(hash(stateVarId, mappingKey))
- stateValue
- ownerPublicKey
- Salt)
Transpilation Steps

1. Syntax verification
2. Dedecoration
3. Solc compilation
4. Redecoration
5. Generate Abstract Syntax Tree
   a. Circuits
   b. Shield contract
   c. Orchestrator
   d. Test
6. Code generation
zApp Architecture

**EVM**
- **Shield.sol**
  - verification-fn1 {}
  - verification-fn2 {}
  - verification-fn3 {}
- **SNARKVerifier.sol**
  - verification-key-fn1
  - verification-key-fn2
  - verification-key-fn3
- **TimberTree.sol**
  - Root

**Orchestrator**
- **API**
- **Setup for Circuits**
- **Deploy contracts**
  - User commitments
  - Keys

**App.zol**
- fn1 {}
- fn2 {}
- fn3 {}

**Zokrates**
- proving-key-fn1
- proving-key-fn2
- proving-key-fn3

**Timber**
- commitment merkle tree
contract Invoice {

    secret mapping(address => uint256) invoices;
    address contractOwner;

    function addInvoice(secret address owner, secret uint256 amount) public {
        require(invoices[owner] == 0);
        unknown invoices[owner] += amount;
    }

    function payInvoice(secret uint256 amount, secret address owner) public {
        require(msg.sender == contractOwner);
        // imagine some payment here
        invoices[owner] -= amount;
    }
}
Calling ./zappify on Invoice.zol

- **App.zol**
  - addInvoice {} 
  - payInvoice {} 

- **EVM**
  - **Shield.sol**
    - addInvoice {}
    - payInvoice {}
  - **SNARKVerifier.sol**
    - verification-key-
    - addInvoice
    - verification-key-
    - payInvoice
  - **TimberTree.sol**
    - Root
    - 0

- **Orchestrator**
  - addInvoiceAPI
  - payInvoiceAPI

- **Zokrates**
  - proving-key-addInvoice
  - proving-key-payInvoice

- **Timber**
  - invoice merkle tree
When user calls addInvoiceAPI
Thank you!

Chaitanya Konda
Assistant Director, Ernst & Young LLP
EY Global Blockchain
ckonda@uk.ey.com