Privacy

Stealth addresses, mixers and Privacy Pools · Session 16 · 16 November 2026

Where we left off

Every transaction, balance and contract call in this part of the course was public. This session: what that exposes, which tools hide part of it, and what they cost.

Part	Subject
1	What a public chain reveals
2	A new address for each payment, and stealth addresses
3	A mixer, written in class: the encoding error
4	Zero-knowledge proofs, in brief
5	Tornado Cash
6	Privacy composed with compliance: Privacy Pools
7	A credential for the course badges

What an address reveals

An observer learns	From
Balance and holdings	the state: ether and every token
Complete history	every transfer, in order, with its time
Counterparties	who funded the address, whom it pays
Behaviour	which contracts, how often, at what hour
First funding	the address that paid for its first transaction

Example: vault 0x170B…1650, every call to it, including the failed ones.

From an address to a person

Link	How it works
Exchange account	the withdrawal address is tied to a verified customer
Naming	ENS names, social profiles, donation pages, airdrop claims
Reuse	the same address in every application
Funding	the first transfer, or the gas, came from a known address
Patterns	amounts, hours, contracts used

One link identifies the address, and with it **all its history**, backwards and forwards.

Why it matters

Who	What a public ledger exposes
Individuals	savings, salary and spending; holdings that can make a person a target
Firms	supplier prices, treasury movements, payroll, a strategy others can copy
Voting	an election on a blockchain: anyone could pair each voter with a ballot

A result can be checked by anyone **without** exposing who voted for what: that needs more than a public ledger.

Three properties

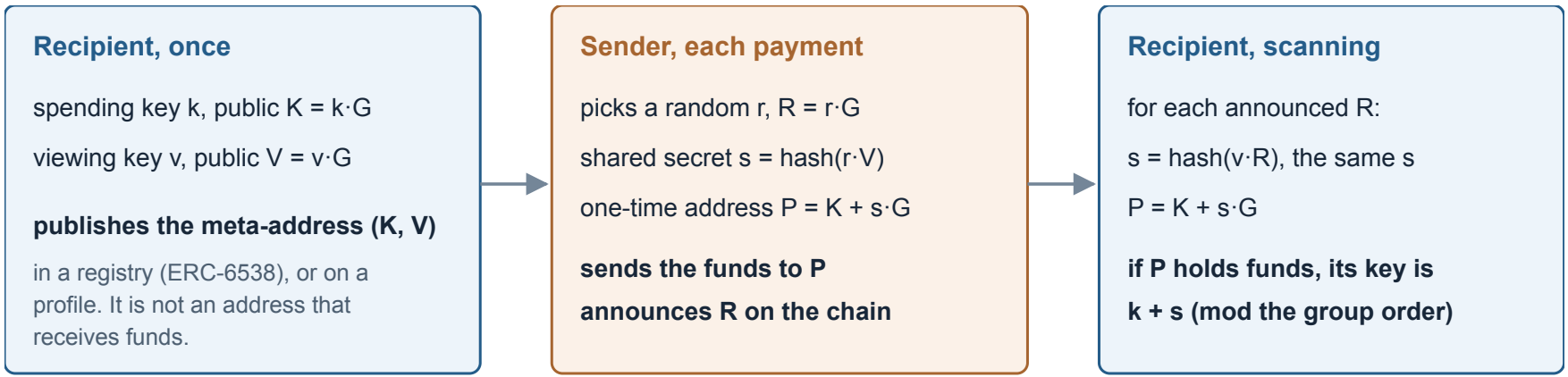
Property	Meaning	Tool
Confidentiality of amounts	the value moved is hidden	not covered here
Anonymity of parties	who sends, who receives	fresh and stealth addresses
Unlinkability	a payment in cannot be tied to a payment out	mixers, Privacy Pools

A new address for each payment

Gain	Limit
a fresh address has no history	the payer knows the address
the shop’s other payments stay unlinked	the funds must be consolidated, and the structure shows
addresses are not scarce	each customer must be given a new address: interactivity

Interactivity rules it out for a static QR code or a donation page.

Stealth addresses



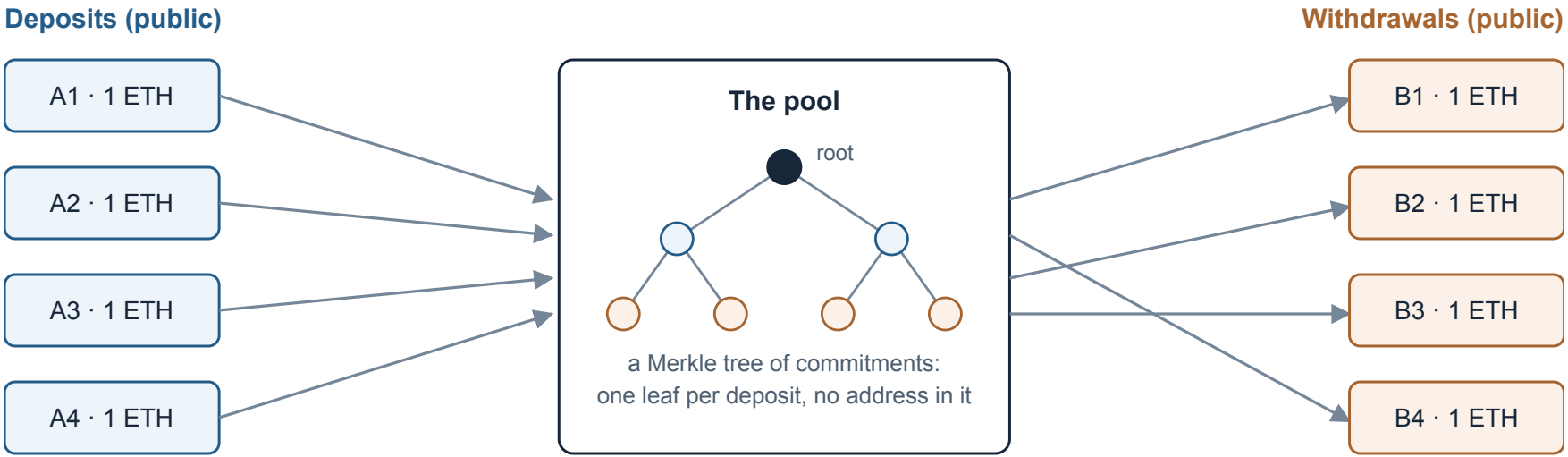
An observer sees

a payment to P, and a number R. Nothing ties P to K or V, so nothing ties P to the recipient's other addresses.

What a stealth address leaves visible

- The **amount**, and the payer, who can follow the address afterwards.
- The **gas problem**: the address holds tokens but no ether, so moving them needs gas, and funding it from a known address links it again.
- Someone else can pay the gas for a signed message and be repaid in another token (paymaster, session 15).
- The **sender** is not hidden: ring signatures, as in Monero, hide it among a set of keys.

A mixer: a pool of deposits



Which Ai paid which Bj is not on the chain, if the pool is large and the amounts are equal.

A hash lock

Deposit under a hash; withdraw by revealing a value that has that hash.

```
mapping(bytes32 h => uint256 amount) public pot;     // hash => locked ether

function deposit(bytes32 h) external payable { pot(h) += msg.value; }

function withdraw(bytes32 secret) external {
    bytes32 h = keccak256(abi.encode(secret));       // the contract hashes 32 bytes
    uint256 amount = pot(h);
    if (amount == 0) revert NothingToWithdraw();     // no pot under this hash
    pot(h) = 0;
    (bool ok, ) = msg.sender.call{value: amount}("");
}

function hashOf(bytes32 secret) external pure returns (bytes32) {
    return keccak256(abi.encode(secret));            // the same computation, for callers
}
```

Source: HashLock 0x247A…9084, deployed in class.

What `withdraw` hashes

- `secret` is a `bytes32`: 32 bytes, passed as they are.
- `abi.encode` of a `bytes32` is those same 32 bytes: no length, no padding added.
- So the contract looks for a pot under **keccak256 of the 32 bytes**.

A **text** is not 32 bytes: "12345" is 5 bytes. Hashing the text gives a different value from hashing the 32-byte value that holds it.

Two hashes, in order

Step	Input	Output	Used as
1	the text `12345`, as UTF-8	`0x1841d653…10e6`	the **secret**, passed to `withdraw`
2	that output, as **hex** (the 32 bytes)	`0x7145c408…a78c`	the **hash**, passed to `deposit`

Check: `hashOf(0x1841d653…10e6)` on the contract returns `0x7145c408…a78c`.

Tool used in class: emn178 keccak-256, with the input type set to UTF-8 in step 1 and to **Hex** in step 2.

What each choice leads to

What is deposited, and how it is withdrawn	Hash computed	Result
step 2 with **hex** input, then `withdraw(0x1841…10e6)`	`0x7145c408…a78c`	works
step 1 only: the text hashed once, used as the deposit hash	`0x1841d653…10e6`	no `secret` hashes to it
step 2 with **UTF-8** input, characters without `0x`	`0xa8bed625…aa99`	no `secret` hashes to it
step 2 with **UTF-8** input, characters with `0x`	`0xd8134b5d…6e14`	no `secret` hashes to it
the text padded to 32 bytes, `0x3132333435` + zeros, as the secret	`hashOf` gives `0x29f31333…6750`	works only if deposited under this hash

A deposit under a hash that **no 32-byte value** produces cannot be withdrawn: the contract only accepts `secret` values.

A deposit that cannot be withdrawn

Read on the contract	Value
`pot(0x1841d653…10e6)`, the hash that was deposited	0.1 ETH
`pot(0x7145c408…a78c)`, the hash a withdrawal looks up	0

The deposit was made under the hash of the text. The contract needs a `secret` whose hash is `0x1841d653…10e6`; finding one means inverting keccak-256. The contract has **no administrator and no rescue function**.

Check: read `pot` on Etherscan, and the deposit transaction.

Test the whole flow before production

What was tested	What was not
the deposit: the transaction went through	the withdrawal, with the secret as the contract reads it
the hash, computed with an external tool	the hash, computed as the contract computes it

- A **round trip** with a small amount first: deposit, then withdraw, with the steps used for the real amount.
- Values from the contract's own function (`hashOf`), not from a re-implementation of its encoding.

On a test network the cost is a few minutes. On mainnet it is **permanent**.

Hardening the lock

Problem	Change
the secret is public in the withdrawal transaction, and anyone can copy it	**commit** to the withdrawal first (a hash of address and secret); withdraw only some blocks later
a deposit nobody withdraws is lost	**refund** to the depositor after a time limit
two hashes, easy to get wrong	**one** hash: deposit `keccak256(text)`, withdraw with the text
a short secret can be guessed	use a long random text

A version with these changes, written and verified after the session: HashLockTest 0x6bA6…329B. The commit delay limits front-running; it does not remove it.

Why a hash lock is not a mixer

What is public	Consequence
the same hash `h` in the deposit and in the withdrawal	the withdrawal can be paired with its deposit
the secret, once revealed in the transaction	whoever sees it first can use it (front-running)
`withdrawTo` pays another address	the **recipient** changes, the link through `h` does not

Hashing the revealed secret gives the commitment: input and output are **linked**.

What a mixer needs

Condition	Why
Equal amounts	a unique amount is a fingerprint that pairs a deposit with a withdrawal
Many users	the **anonymity set** is the deposits that could be yours: one deposit means none
Time between deposit and withdrawal	an immediate withdrawal pairs them by timing
A withdrawal that does not name its deposit	otherwise the contract itself reveals the link

The last condition is the hard one: the contract must pay **only the rightful owner** without knowing which deposit is theirs.

Proving without revealing

A **zero-knowledge proof** convinces a verifier that a statement is true, and nothing else.

Property	In words
Completeness	an honest prover with a true statement convinces the verifier
Soundness	a prover with a false statement cannot, except with negligible probability
Zero knowledge	the verifier learns nothing beyond the truth of the statement

Applied to the hash lock: "I know a secret whose hash is **one of the deposits**", without saying which.

What the mixer proves

Public	Private
the Merkle root of all commitments	the secret and the nullifier chosen at deposit
the nullifier hash	the Merkle path of the commitment
the recipient address	

Statement: a commitment built from this secret and nullifier is a leaf under this root, and the nullifier hash is the hash of this nullifier.

The contract checks the proof, checks that the nullifier hash was never used, and pays the recipient.

Tornado Cash

What	a mixer on Ethereum, launched in 2019
Pools	fixed amounts of ether (0.1, 1, 10 and 100) and some tokens
Control	immutable contracts; a governance token and relayers around them
Use	privacy for ordinary payments, and laundering of stolen funds

A pool that treats every deposit alike cannot tell the two apart, and **neither can its users**.

Sanctions

When	What happened
Aug 2022	the US Treasury (OFAC) lists the mixer’s contract addresses: US persons may not use them
Reason given	some deposits came from illicit sources, so a withdrawal could be from one of them
Effect	services block the addresses; the web front end and the code repository are taken down

Privacy that cannot be separated from **who uses it** collides with compliance.

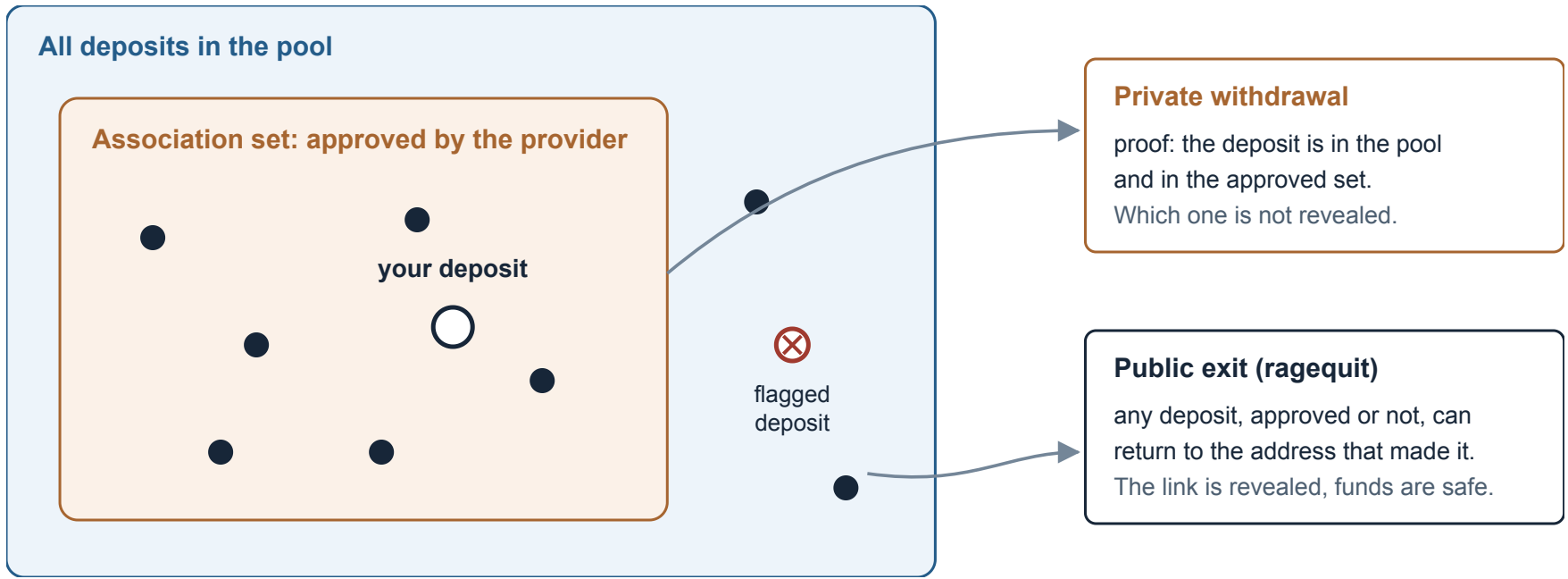
Proving membership in a subset

Instead of "my deposit is one of **all** deposits", prove "my deposit is one of the deposits in a set **S** that someone vouches for".

- Privacy holds **among the members** of S.
- Membership in S is the statement a third party can verify.
- Who chooses S, and by what rule, is an explicit, public parameter.

Proposal: Buterin, Illum, Nadler, Schär and Soleimani, *Blockchain privacy and regulatory compliance: towards a practical equilibrium* (2023).

Association sets



Privacy Pools

Step	What happens	What is visible
Deposit	a public transaction puts ether or a token in the pool; a commitment joins the tree	the address, the amount
Vetting	the provider screens the deposit and publishes a Merkle tree of approved deposits	one decision per deposit
Withdrawal	a proof: the deposit is in the pool **and** in the approved set; a relayer sends the funds to a new address	the recipient, the amount; not which deposit
Public exit	a deposit not approved, or by choice, returns to the depositing address	the link

Project: 0xbow. Live on Ethereum mainnet; the repository is open (Apache-2.0).

What the provider can and cannot do

Can	Cannot
decide which deposits are in the set	take the funds: anyone can exit publicly
remove a deposit from a later list	link a withdrawal to a deposit
be replaced by another provider with another rule	see the secrets behind a commitment

Costs:

- an **error** of the provider excludes an honest user from private withdrawals;
- the **anonymity set** shrinks to the approved deposits that match the amount;
- the deposit itself is public, from a known address.

Compliance as a parameter

Design	Who is in the set	Who decides
Tornado Cash	every deposit	nobody
Privacy Pools	deposits that pass screening	the provider
A set of verified customers	depositors holding a credential	the issuer of the credential
A set for one counterparty	whatever that counterparty requires	the counterparty

The proof is the same; **the statement is chosen**.

A credential for the course badges

The badges are shown on a web page. If the page goes away, nothing shows that a badge was earned.

A **credential** is a statement signed by the issuer: this address earned these badges.

1. The holder downloads it and keeps it.
2. The holder can show it to anyone, and prove control of the address by signing a challenge.
3. Anyone can check the issuer’s signature without the course site.
4. A separate step can link the credential to a university account, through the university email.

The properties compared

	Hides the receiver	Breaks in–out link	Hides amounts	Needs trust in	Compliance
Plain address	no	no	no	nobody	by inspection
Fresh address	until funded	no	no	nobody	by inspection
Stealth address	yes	no	no	nobody	by inspection
Mixer	yes	yes	no	a setup	none
Privacy Pool	yes	yes, within the set	no	a setup, a provider	by the set

What stays open

- **Amounts**: confidential transfers and encrypted balances, with their own trade-offs.
- **Metadata**: the node a wallet calls, the network address, the browser: privacy is not only on chain.
- **Law**: rules on anonymous accounts and on assets that hide the holder differ by jurisdiction and are changing.
- **Design**: who should run an association set, and what an excluded user can do.

Where this part of the course ends

Session	Subject
8–10	the world computer; smart contracts; a shared vault
12–13	token standards; selling a token, markets
14–15	keys and recovery; smart accounts and Safe
16	privacy, and privacy composed with compliance

Each step chose a **trade-off** between control, recovery, visibility and trust.