BTQ Core / Community Note

A Field Guide to BTQ Transaction Types

Live examples on the public testnet explorer

Oscar Chambers · Barney Chambers

Network: BTQ public testnet (`v0.4.2-testnet`)

Date: 28 July 2026

Explorer: https://explorer.bitcoinquantum.com

Abstract

BTQ has two families of addresses and spends: the usual ECDSA types people know from Bitcoin (for compatibility), and Dilithium types for post-quantum custody. This note walks through those choices. For each type we cover what it is, why we included it, why it is useful, and point to a real confirmed testnet transaction so readers can open the explorer and inspect the on-chain details (inputs, outputs, script types, and witnesses) for themselves.

Keywords: BTQ, Dilithium, P2MR, ECDSA, Taproot, multisig, block explorer, post-quantum signatures.

1 Why this exists

We published these examples so people can *see* the types we chose to include, understand why they are there, and learn by clicking through live transactions.

> Open any explorer link below, then look at outputs, witnesses, and sizes.
> That is what each type looks like on-chain.

For each type you will find:

1. **What it is**: structure of the address/script and how a spend is authorized;
2. **Why we included it**: why it is in BTQ at all;
3. **Why it is useful**: who benefits and in which situations;
4. **A live example**: a mined testnet transaction to browse.

2 Quick catalogue

Table 1: Types with live explorer examples.

Type	Why it's here	Explorer
ECDSA legacy P2PKH	Classic Bitcoin compatibility	`8491a822...`
ECDSA p2sh-segwit	Bridge for older SegWit wallets	`2c9af353...`
ECDSA bech32	Default modern SegWit receive	`559e061b...`
ECDSA bech32m / Taproot	Advanced ECDSA script trees	`5ed561a6...`
ECDSA wsh 2-of-3	Shared ECDSA custody	`7b07314d...`
Dilithium P2MR single-key	Standard post-quantum receive	`18bd0dc8...`
P2MR `OP_TRUE` leaf	Demo of the P2MR script-path layout	`a0a068b8...`
Dilithium P2MR multisig 2-of-3	Dilithium multisig custody	`3301fec1...`
Dilithium P2MR threshold 2-of-3	Another Dilithium threshold script style	`d1b26028...`
Legacy Dilithium Base58	Older Dilithium format (Section 5.1)	`665eb4ac...`

3 ECDSA family: compatibility and everyday use

We keep the usual ECDSA address types. Dilithium is where we want custody to go over time, but most wallets, exchanges, hardware devices, and operators still speak ECDSA. Supporting these types means people can move value on BTQ today with tools they already know, while Dilithium can grow without everyone having to switch overnight.

Oldest to newest:

1. **Legacy**: oldest, most widely recognized, least efficient;
2. **P2SH-SegWit**: SegWit fees, but the address still looks like old P2SH;
3. **Bech32**: modern default for single-key ECDSA;
4. **Taproot**: modern default for more complex ECDSA policies;
5. **wsh multisig**: shared custody with ECDSA.

3.1 Legacy P2PKH

What it is. Pay-to-PubKey-Hash is the original Bitcoin payment style most people still picture when they hear "crypto address." The address encodes a hash of an ECDSA public key (Base58Check). On testnet these often look like `m.../n...` addresses.

On-chain, the output script is the familiar `OP_DUP OP_HASH160 <20-byte-hash> OP_EQUALVERIFY OP_CHECKSIG`. To spend, the owner reveals the full public key and an ECDSA signature in the **scriptSig** (not in a SegWit witness). Nodes hash the public key, check it matches the output hash, then verify the signature.

Why we include it. A lot of docs, older wallets, exchange deposit code, and muscle memory still assume this shape. Dropping it would make migration from Bitcoin-like tooling harder, and would make basic interoperability testing harder too. Nobody would call it the best receive type today, but it is still the classic one people expect. If we want BTQ to feel familiar to Bitcoin users, we should still handle it.

Why it is useful.

- Easiest idea for newcomers ("address in, signature out");
- Works with the broadest range of older software;
- Useful reference when comparing fees and sizes against SegWit and Dilithium;
- Simple fallback deposit address when you need one.

Explore: `8491a822...104d629f`

Look for a `pubkeyhash` output and authorization in scriptSig rather than a witness stack.

3.2 P2SH-SegWit

What it is. P2SH-SegWit wraps a SegWit program inside a Pay-to-Script-Hash address. The address itself looks like older P2SH, but when spent correctly the heavy lifting happens in the witness, and fee accounting uses SegWit weight rules. Older software can often *send to* the address because it only sees P2SH; newer software can *spend from* it efficiently because it knows SegWit.

Why we include it. This is the transition format from the SegWit rollout. A lot of wallets and services adopted SegWit first as nested P2SH so the other side did not have to understand native bech32 yet. Supporting it means BTQ still works with that generation of software, without forcing everyone straight to native SegWit or Taproot. It is also a good reminder that the address someone pastes is not always the whole story of how the spend is authorized in the transaction.

Why it is useful.

- SegWit fee efficiency while the address still looks like ordinary P2SH, so older senders accept it;
- Useful when the sender's wallet is old and the receiver's is newer;
- Clear example of how upgrades can ship in stages;
- Still common in the wild, so supporting it avoids weird deposit failures.

Explore: `2c9af353...0594e330`

Compare with legacy P2PKH: more authorization moves into the witness path.

3.3 Bech32 (P2WPKH)

What it is. Native SegWit version 0 pay-to-witness-public-key-hash. On BTQ testnet these addresses look like `tbtq1q...`. The output is a short witness program: version `0` plus a 20-byte

public-key hash. Spending puts the public key and ECDSA signature in the **witness**. Bech32 encoding adds better checksums than Base58, so typos are more likely to be caught before a send.

Why we include it. This is the modern default for ordinary single-key ECDSA receives. If someone asks what ECDSA address a wallet should generate for most payments, bech32 P2WPKH is the answer. Including it keeps BTQ in line with how Bitcoin wallets work today, and gives people who care about fees a compact option that does not need Taproot.

Why it is useful.

- Lower fees than legacy for the same transfer;
- Safer addresses and better typo detection;
- Simple single-signer custody without multisig overhead;
- Best ECDSA example to put next to Dilithium P2MR later.

Explore: `559e061b...b1d54e5`

Look for `witness_v0_keyhash` and a compact witness stack.

3.4 Bech32m / Taproot (P2TR)

What it is. Taproot (witness version 1, bech32m) commits to a public key that can represent either a **key-path** spend (one signature that looks like a simple key) or a **script-path** spend (reveal a leaf from a Merkle tree of scripts when more complex conditions are needed). If participants can cooperate via key-path, the chain need not advertise all backup scripts.

Why we include it. Taproot is where more advanced ECDSA scripts live on Bitcoin-like chains. We support it so you are not stuck between simple P2WPKH and awkward older script formats when you need complicated spending rules. It also makes P2MR easier later: both use tree-structured script commitments, even though Dilithium lives in P2MR rather than BIP341 Taproot leaves.

Why it is useful.

- Better privacy defaults when you have complicated internal spending rules;
- You can keep backup spending conditions without showing them every time you spend;
- Uses the newer bech32m address format;
- Once you understand Taproot trees, P2MR Dilithium trees are easier.

Explore: `5ed561a6...d489a22`

Check whether the spend is key-path or script-path.

3.5 ECDSA wsh 2-of-3 multisig

What it is. A native SegWit pay-to-witness-script-hash (`wsh`) output whose witness script is a 2-of-3 ECDSA multisig. Three public keys are committed; any two corresponding signatures authorize the spend. In the explorer you should see a witness that includes the witness script and two signatures.

Why we include it. Shared custody did not go away when SegWit arrived. Treasuries, exchanges, mining ops, and multi-person projects still need "no single laptop can drain the funds." ECDSA multisig is how most people do that today. A live 2-of-3 `wsh` example also sets up a clean comparison with Dilithium multisig later: same 2-of-3 rule, different signature algorithm.

Why it is useful.

- Lose or compromise one key without losing funds;
- Two people must sign before money moves;
- Matches how existing multisig ops already work;
- Easy example if you are explaining BTQ custody to someone.

Explore: `7b07314d...e1fee94`

Inspect the witness script policy and confirm two signatures are present.

4 Dilithium family: post-quantum custody via P2MR

Dilithium is BTQ's post-quantum signature system. The tradeoff is large public keys and signatures, in exchange for resistance to the attacks that would threaten ECDSA if large quantum computers arrive.

We do not sprinkle Dilithium across every script type. It lives in **P2MR** (Pay-to-Merkle-Root, witness version 2): an output that commits to the Merkle root of a script tree. Leaves may be a single Dilithium key check, Dilithium multisig, threshold accumulators, or other script policies (including demo leaves like `OP_TRUE`).

Addresses look like `tbtq1z...` on testnet. That prefix should stand out: Dilithium should be obvious at a glance, not easy to mix up with ECDSA `tbtq1q...`.

4.1 Dilithium P2MR single-key

What it is. The normal Dilithium receive address from `getnewdilithiumaddress`. Under the hood this is usually a P2MR output whose tree has a leaf that checks one Dilithium signature against one Dilithium public key. Spending reveals that leaf, the control block proving membership in the committed Merkle root, and a Dilithium signature. In the explorer, the first thing you notice is size: Dilithium witnesses dwarf ECDSA ones. With the parameters we use today, that is simply what post-quantum signatures look like.

Why we include it. This is the Dilithium payment address we want wallets, mining pools, and users to use. If Dilithium is a big part of why BTQ exists, there should be one clear answer to "send me Dilithium-secured coins," not a pile of experimental formats. Single-key P2MR is that answer, and the right payout target for mining.

Why it is useful.

- Quantum-resistant coins for normal payments and savings;
- One Dilithium address type to teach and display;
- A real Dilithium example for wallet and fee work to measure against;
- One post-quantum deposit format mining and exchanges can agree on.

Explore: `18bd0dc8...09f4c2`

Open next to the bech32 ECDSA example: same job, very different witness size.

4.2 P2MR `OP_TRUE` leaf

What it is. A P2MR output whose spending leaf is trivial on purpose (`OP_TRUE`). We use it to show the P2MR spend path by itself: Merkle leaf, control block, script-path witness layout, without a giant Dilithium signature filling the screen. Anyone who can build the correct script-path witness can spend such an output. Fine as a demo UTXO, but do not use it as a wallet receive address for real funds.

Why we include it. P2MR is new to a lot of readers. If the first Dilithium example mixes a new output type, a Merkle proof, and a kilobyte-scale signature, it is hard to tell what is what. The `OP_TRUE` leaf splits that apart: first learn how a P2MR script-path spend is structured, then look at Dilithium leaves that use the same structure with real authorization. We put a live example on-chain so people can inspect the bare structure.

Why it is useful.

- Easiest place to start learning control blocks and leaf reveals;
- Makes Dilithium multisig examples easier to read afterward;
- Useful if you are writing an explorer, indexer, or wallet UI that has to show P2MR paths;
- Shows P2MR is a general script tree, not only a Dilithium wrapper.

Explore: `a0a068b8...218daa`

Focus on witness structure (leaf script + control block) rather than signature cryptography.

4.3 Dilithium P2MR `CHECKMULTISIG` 2-of-3

What it is. A P2MR leaf that uses `OP_CHECKMULTISIGDILITHIUM` with a classic multisig layout: commit three Dilithium public keys, require two signatures. In practice it is the same approval rule

as ECDSA 2-of-3. Every signature is Dilithium, and the whole policy sits inside a P2MR script path rather than a SegWit v0 `wsh`. Expect a large transaction.

Why we include it. Real custody usually needs more than one key, so organizations want threshold approval. Native Dilithium multisig inside P2MR means shared custody does not have to fall back to ECDSA the moment a second signer shows up. We use a concrete 2-of-3 because it is the pattern people already know.

Why it is useful.

- Dilithium-secured treasuries without a single point of key failure;
- Same signing process people already know from Bitcoin multisig;
- Easy to compare with the ECDSA `wsh` 2-of-3 example;
- Useful for exchange cold wallets, foundation wallets, and protocol treasuries.

Explore: `3301fec1...b75ab9c`

Compare with the ECDSA wsh example: policy similarity, cryptographic difference.

4.4 Dilithium P2MR threshold accumulator 2-of-3

What it is. An alternative way to express "2 of 3 Dilithium keys must approve" without using the native CHECKMULTISIG opcode layout. The leaf accumulates per-key signature checks (verify Dilithium sig or accept an empty contribution as zero, then add, then require the total to meet threshold m), built from `OP_CHECKSIGDILITHIUM` with `OP_ADD`/compare logic similar to BIP342-style accumulators. Any qualifying subset can sign (keys 1+2, 1+3, or 2+3).

Why we include it. CHECKMULTISIG's old stack rules don't fit every policy. Accumulators are more flexible for script authors and closer to how people write modern tapscript thresholds. A live accumulator 2-of-3 on testnet shows that P2MR Dilithium is not stuck with one multisig style. You can pick the leaf that matches your wallet and policy tools.

Why it is useful.

- Any valid subset of signers can approve, as long as you hit the threshold;
- Easier for some wallets and policy tools to generate;
- Useful contrast: two on-chain ways to express the same 2-of-3 rule;
- Starting point for richer Dilithium policies.

Explore: `d1b26028...7415c4`

Open beside the CHECKMULTISIG example and compare leaf scripts for the same 2-of-3 idea.

5 Design note: why P2MR is the Dilithium home

We put Dilithium in one place for a few practical reasons:

1. **One address type.** One Dilithium look (`tbtq1z...`), not a pile of similar-looking formats.
2. **Room for scripts.** Single-key, multisig, and custom leaves live in the same tree model.
3. **Cleaner upgrades.** Dilithium opcodes belong in P2MR tapscript, not mixed into witness-v0 shapes that look like ECDSA.
4. **Easier ops.** Mining, wallets, and explorers can all treat one Dilithium receive type as the default.

5.1 Historical example: Legacy Dilithium Base58

What it is. An earlier Dilithium address format from before P2MR was finalized: Base58 plus a `dilithium_pubkeyhash`-style script using `OP_CHECKSIGDILITHIUM` outside the P2MR tree.

Why we show an example. Testnet still has outputs like this. Showing one live spend makes the history clear: here is what early Dilithium looked like, and here is why P2MR is what we teach now.

How to think about it. Treat this as history: useful for seeing where we came from, but new wallets and mining products should use P2MR instead.

Explore: `665eb4ac...c14ba4`

Compare with the P2MR Dilithium single-key example.

> Dilithium bech32 / witness-v0 was an early experiment we turned off, because Dilithium and ECDSA witness programs were too easy to mix up. There is no live example to browse for that path.

6 A short path through the explorer

1. Open an **ECDSA bech32** spend and a **Dilithium P2MR** spend side by side: same "send coins" idea, very different witness weight.
2. Open the **ECDSA 2-of-3** and **Dilithium CHECKMULTISIG 2-of-3**: same custody idea, different cryptography.
3. Open the **`OP_TRUE` leaf** then a Dilithium leaf spend: first the bare P2MR structure, then real Dilithium authorization.
4. Open **CHECKMULTISIG Dilithium** and **threshold accumulator Dilithium**: same 2-of-3 rule, two different script styles.

A Full transaction identifiers

Table 2: Complete transaction identifiers for the examples above.

Type	Full txid
ECDSA legacy P2PKH	`8491a822c29e9a3a905f50e20e6c13465f36247842dce7afb0c188ca104d629f`
ECDSA p2sh-segwit	`2c9af353c0045e61696af9b818c378074fd0cc552182f7be70c77bb10594e330`
ECDSA bech32	`559e061bb24e6ee1c85c132e319223bf9bd9aace9420d65a995d44e1bb1d54e5`
ECDSA bech32m / Taproot	`5ed561a69a31c55f73fa566bc1f060539e71167522a6fd9680046d93ed489a22`
ECDSA wsh 2-of-3	`7b07314d73b0f12782e677c46a668c1abb059054ef1a1b4ee58462586e1fee94`
Dilithium P2MR single-key	`18bd0dc86094a8390583fb155313cefae0427dd766fe01c9a328133e4209f4c2`
P2MR `OP_TRUE` leaf	`a0a068b8a7a02a35f941a354db9fbc08a60cc8d7dd84cd9923b09c633b218daa`
Dilithium P2MR multisig 2-of-3	`3301fec1f5899252cf582c64c0f92f877755ab0f29783bb7001e422fdb75ab9c`
Dilithium P2MR threshold 2-of-3	`d1b2602835b76ee714189f70d3a91f5d32fd003ebe283370970a6f3c1f7415c4`
Legacy Dilithium Base58	`665eb4ac94212f050a3c5cb8a5eba30666e3a9216611d0f21854fe37a7c14ba4`

Oscar Chambers · Barney Chambers · BTQ Core

Explorer: https://explorer.bitcoinquantum.com