



		Unlocking Script / scriptSig / Witness	Locking Script / scriptPubKey / Witness Program
Legacy	P2PKH	<p><b>&lt;Signature&gt; &lt;PubKey&gt;</b></p> <p>written by the payee (payment destination) of a previous transaction when it becomes the payer (payment source) of a new transaction in which the UTxO has to be spent</p>	<p><b>DUP HASH160 &lt;PubKeyHash&gt; EQUALVERIFY CHECKSIG</b></p> <p>where:</p> <p><math>\text{PubKeyHash} = \text{HASH160}(\text{PubKey})</math></p> <p>↓</p> <p>generates <b>1* address</b> via Base58Check encoding with 0x00 prefix</p>
	P2SH	<p><b>RedeemingData &lt;RedeemScript&gt;</b></p> <p>↓</p> <p>Data compliant with BIP16 prescriptions and satisfying RedeemScript (e.g., enough stack-PUSHed signatures in multisignature contract)</p> <p>↓</p> <p>BIP16's two stages for P2SH:</p> <ol style="list-style-type: none"> <li>1) POP the top stack data and verify it against the locking script;</li> <li>2) if previous check is successful, deserialize the POPped data and use it as new locking script for the remaining part of the unlocking script</li> </ol>	<p><b>HASH160 &lt;RedeemScriptHash&gt; EQUAL</b></p> <p>where:</p> <p><math>\text{RedeemScriptHash} = \text{HASH160}(\text{RedeemScript})</math></p> <p>↓</p> <p>generates <b>3* address</b> via Base58Check encoding with 0x05 prefix</p> <p>by checking if address begin with "1" or "3", the payer can build the right locking script for payee's address</p>
Segwit Native	P2WPKH	<p><i>empty scriptSig field</i> (“spendable by everyone”)</p> <p><b>&lt;Signature&gt; &lt;CompressedPubKey&gt;</b></p> <p>as P2PKH, but everything in “segregated” structure</p>	<p><b>0 &lt;PubKeyHash&gt;</b></p> <p>↓</p> <p>Segwit version (Q3 2019)</p> <p>↓</p> <p>they generate <b>bc1* address</b> via Bech32 encoding</p> <p>where:</p> <p><math>\text{PubKeyHash} = \text{HASH160}(\text{CompressedPubKey})</math></p> <p>uncompressed keys are ok in legacy cases, but nonstandard in Segwit</p>
	P2WSH	<p><i>empty scriptSig field</i> (“spendable by everyone”)</p> <p><b>RedeemingData &lt;RedeemScript&gt;</b></p> <p>as P2SH, but everything in “segregated” structure</p>	<p><b>0 &lt;RedeemScriptHash&gt;</b></p> <p>↓</p> <p>Segwit version (Q3 2019)</p> <p>↓</p> <p>they generate <b>bc1* address</b> via Bech32 encoding</p> <p>where:</p> <p><math>\text{RedeemScriptHash} = \text{SHA256}(\text{RedeemScript})</math></p> <p>different sizes (for “bc1” addresses too), but same locking scripts form</p>
Segwit Compatibility	P2WPKH inside P2SH	<p><b>&lt;0 &lt;PubKeyHash&gt;&gt;</b></p> <p>BIP16 2<sup>nd</sup> stage is P2WPKH ↕</p> <p><b>&lt;Signature&gt; &lt;CompressedPubKey&gt;</b></p> <p>in “segregated” structure</p>	<p><b>HASH160 &lt;RedeemScriptHash&gt; EQUAL</b></p> <p>where:</p> <p><math>\text{RedeemScriptHash} = \text{HASH160}(\text{RedeemScript})</math></p> <p><math>\text{RedeemScript} = 0 \text{ &lt;PubKeyHash&gt;}</math></p> <p><math>\text{PubKeyHash} = \text{HASH160}(\text{CompressedPubKey})</math></p> <p>P2SH</p> <p>P2WPKH</p>
	P2WSH inside P2SH	<p><b>&lt;0 &lt;ActualRedeemScriptHash&gt;&gt;</b></p> <p>BIP16 2<sup>nd</sup> stage is P2WSH ↕</p> <p><b>RedeemingData &lt;ActualRedeemScript&gt;</b></p> <p>in “segregated” structure</p>	<p><b>HASH160 &lt;RedeemScriptHash&gt; EQUAL</b></p> <p>where:</p> <p><math>\text{RedeemScriptHash} = \text{HASH160}(\text{RedeemScript})</math></p> <p><math>\text{RedeemScript} = 0 \text{ &lt;ActualRedeemScriptHash&gt;}</math></p> <p><math>\text{ActualRedeemScriptHash} = \text{SHA256}(\text{ActualRedeemScript})</math></p> <p>P2SH</p> <p>P2WSH</p>
Notes		<p><b>HASH160(x) = RIPEMD160( SHA256(x) )</b> generates 20 bytes hash</p> <p><b>SHA256(x)</b> generates 32 bytes hash</p> <p><b>&lt;x&gt;</b> transaction script operator that PUSHes in the stack the x data by means of an opcode declaring x's size in bytes</p> <p>IMPORTANT: if the hashed or pushed x is a script, the actual data being processed is the script serialization</p>	
(not exhaustive) Credits		<p>Andreas M. Antonopoulos's <i>Mastering Bitcoin 2<sup>nd</sup> Ed.</i> – O'Reilly (especially chapters 4, 6, 7, appendixes B and D)</p> <p>Jimmy Song's <i>Understanding Segwit Block Size</i> on Medium</p> <p>Bitcoin Improvement Proposals : <a href="#">BIP 16</a>, <a href="#">BIP 141</a>, <a href="#">BIP 173</a>, ...</p> <p>Greg Walker's <i>P2SH page</i> on his <a href="http://learnmeabitcoin.com">learnmeabitcoin.com</a> website</p> <p>yaoshiang's <i>bitcoin-script-disassembler</i> on GitHub</p>	